Parameter Control for Evolutionary Algorithms

Karafotias, G.

2016

document version
Publisher's PDF, also known as Version of record

VRIJE UNIVERSITEIT

Parameter Control for Evolutionary Algorithms

ACADEMISCH PROEFSCHRIFT

ter verkrijging van graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op woensdag 24 februari 2016 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Georgios M. Karafotias

geboren te Marousi, Griekenland

promotor: prof.dr. A.E. Eiben
copromotor: dr. M. Hoogendoorn

Parameter Control for Evolutionary Algorithms

Cover Art: Recursive Subdivision by Jason Smith

SIKS Dissertation Series No. 2016-10
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Table of Contents

ABSTRACT

This thesis studies the dynamic control of the parameters of evolutionary algorithms, its potential advantages and the methods that can be used to effectively achieve it. Evolutionary algorithms involve several parameters and their settings can have substantial effects on the resulting performance of the algorithm. As a minimum, these parameter must be set to values that will remain static during the run; it is a process that is unavoidable before starting an evolutionary run. In an expanded approach, parameter values can be varied dynamically during a run, i.e. they can be subject to control. Parameter control is mainly supported by the argument that different parameter values are better at different times during an evolutionary run, thus an appropriate control strategy can improve the performance of an algorithm. Though several such strategies have been proposed in literature, there is still no widely applicable control method. In this work, we tackle the problem of parameter control both on a conceptual and on a practical level and in a broad manner. We present a theoretic framework and identify the components of a controller, we attempt to experimentally evaluate the arguments supporting parameter control in general and we present specific algorithmic designs for generic controllers that can be applied to any evolutionary algorithm and control any parameter.

Acknowledgements

The work you are about to read was made possible thanks to the contribution of several people I worked with during my research.

First and foremost, I would like to thank Gusz Eiben for his advice and guidance, for sharing his knowledge and vision, and for mentoring me as a scientist and researcher.

Second, I would like to thank Mark Hoogendoorn for his support, and for all the discussions that many times helped me to overcome obstacles and move forward.

Furthermore, a special thanks goes to my predecessor Selmar Smit for introducing me to the subject, and for the conversations that inspired some of the core concepts in this thesis.

A special thanks also goes to my office-mates, Jacqueline Heinerman and Berend Weel for always offering their perspective, ideas, and support.

I would also like to thank Luís Simões, Evert Haasdijk and Eelco den Heijer for their fruitful ideas and suggestions.

Finally, I would like to thank all the great people I met and worked with at the Computational Intelligence Group.

Part I

Parameter Control: The Field & Theory

1 Introduction

EVOLUTIONARY algorithms (EAs) are a class of powerful black box optimisation and learning algorithms. EAs are inspired by biological evolution in nature. Possible solutions to a problem are encoded in an appropriate representation, a genome that contains all the information about an individual (a candidate solution). They undergo recombination and mutation operations to produce offspring, a process that corresponds to reproduction in biological organisms. Individuals compete for a place in the population and a chance for mating according to a fitness value that is assigned to them by an objective function expressing the optimisation problem at hand; this resembles the selection that organisms go through in nature and the concept of the survival of the fittest. Though EAs are very far from accurately simulating biological evolution, they borrow the basic notions of it and use them to successfully find good solutions to very difficult problems, i.e. problems that are non-linear, multimodal and of very high dimensionality and vast search spaces. EAs have been applied to problems from a wide range of fields, e.g. engineering [60] and software engineering [130], industrial design [220], bioinformatics [99], finance [51] and robotics [216].

The field of evolutionary computing (EC) started with two main types of EAs: the Ge-

netic Algorithm (GA) and the Evolution Strategy (ES) [81]; though they follow the same concepts, they mainly differ in the representation (binary versus real valued). In the following years, a multitude of algorithms have appeared and new categories have been established, e.g. Genetic Programming [166] and Differential Evolution [262]. At the point these lines were written, a multitude of EAs and variants exist making categorisation and labelling difficult; nevertheless, all EAs follow a common abstract design motif. The core process of an EA consists of a loop of offspring creation (through recombination and mutation) and selection (of parents and of survivors). This loop is illustrated in Figure 1.1. Starting with a population of individuals encoding candidate solutions, a pool of parents is selected, the genomes of parents are recombined and the resulting offspring are mutated and then assigned a fitness by the objective function. A selection process chooses the survivors from the combined pool of the previous population and the new offspring.

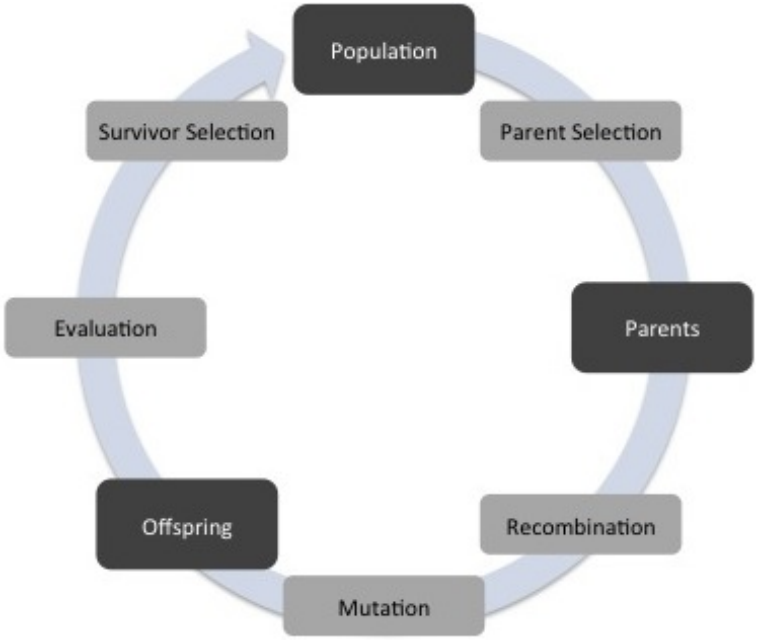


Figure 1.1: *The main loop of an evolutionary algorithm.*

For each of the components of the loop of an EA (parent selection, recombination, mutation, fitness function and survivor selection) there can be several applicable alternatives. Furthermore, every such alternative operator (a specific design implementing one of the loop components) usually includes some numeric values that influence its results. These *parameters* (both choice of operators and settings of their numeric values) define a concrete instance of an EA. Initially, EAs were considered to be robust to their parameter settings, especially GAs, (see for example [62]) but, later, it was demonstrated and now is generally accepted

that parameters can greatly influence the performance of GAs and all EAs in general.[1] One approach in handling the parameter settings of an EA is *parameter control*: varying the parameter values of an EA on-the-fly according to some control mechanism. Parameter control has been a subfield of EC [72] supported by specific arguments (discussed in Section 2.2) and exemplified by several proposed control designs (reviewed in Chapter 3).

The purpose of this thesis is a broad treatment of parameter control both from a theoretical and a practical perspective. The main motivation is that, despite the presence of the parameter control concept in the EC field for the past two decades, it is still an "unsolved problem": it is yet to become a standard practice in the field (or in real world applications for that matter) and the reasons are, mostly, that (i) although certain arguments supporting parameter control are often stated, they still lack a strong foundation with experimental proof that is convincing for the whole EC field (and not only for specific EAs, parameters or cases) and (ii) there are no standard, readily applicable methods one can reach for when employing an (any) evolutionary method. In that direction, we believe that the field could greatly benefit from (i) a coherent, unified framework and theory that can serve as a common basis for researchers to cooperate and (ii) a clear, properly motivated vision for the application (especially in real life) and niche of parameter control. It is the aim of this thesis to contribute to both those objectives. First, we deconstruct entirely the problem of parameter control, inspect the composing subjects, identify the key components and relevant challenges and suggest a framework/view that revises existing views on the matter and can serve as a starting point for further research. Subsequently, we propose a clear vision for the future of the field, geared towards universality and realism. In practical and experimental terms, we present initial applications and results that offer insights about the feasibility and niche of the envisioned practice and make the first steps in assessing the inherent advantages of parameter control in general.

This text is divided into two parts. The first part includes all the theoretical framework and analysis of the field. Chapter 2 presents the basic concepts of parameter control and a theoretical model that explains control and tuning as well as their relation within the overall parameter setting problem; furthermore, it suggests a framework for parameter control from a component viewpoint as well as in the context of Reinforcement Learning. Chapter 3 presents an extensive literature review of the field of parameter control and identifies key trends and challenges that offer valuable insights that inform the directions and choices

[1] Interestingly, the same misconception and subsequent disproof happened for Differential Evolution later on.

taken in the rest of this work. The second part of this thesis is based on the concepts, frameworks and observations of the first part and presents concrete designs for parameter control and experimental work evaluating these designs as well as attempting to answer some more fundamental questions regarding parameter control. Chapter 5 experimentally investigates the isolated effect that parameter variation has on the performance of an EA. Chapters 6 and 7 present and test a number of parameter controllers designed as independent components that can be used as generic plugins for any conventional EA; it also investigates the advantage of these parameter controllers when solving dynamic problems. Chapter 8 departs from the component oriented view and examines a control method that mixes the controller into the evolutionary algorithm and is particularly fit for distributed and physically situated evolutionary systems in dynamic environments. Finally, Chapter 9 presents a summary of this work and important conclusions that resulted from it and suggests key directions for future work.

The work in this thesis has been published in the following articles:

Chapter 3	Parameter Control in Evolutionary Algorithms: Trends and Challenges [155]
Chapter 5	Parameter control: strategy or luck? [153]
	Why parameter control mechanisms should be benchmarked against random variation [154]
Section 6.1	Generic Parameter Control with Reinforcement Learning [152]
Section 6.2	Comparing generic parameter controllers for EAs [157]
Section 6.3	Evaluating Reward Definitions for Parameter Control [156]
Section 7.1	A Generic Approach to Parameter Control [158]
	Tuning Evolutionary Algorithms and Tuning Evolutionary Algorithm Controllers [159]
Section 8.1	It's Fate: A Self-organising Evolutionary Algorithm [32]
Section 8.2	Fate Agent Evolutionary Algorithms with Self-adaptive Mutation [17]

2

Theory & Concepts

As the first step in our study on parameter control, before even discussing the existing work in the field, we are going to present a theoretical model and the relevant concepts. This theory is not an empty formalism for sake of it (or for the sake of appearances); there are three very pragmatic points motivating it's place in this thesis:

– Offer a design model to researchers and practitioners developing new controllers or engaging in experimentation in the field. It will help comprehending the problem better and perhaps allow a more systematic and effective approach to designing new controllers, avoiding to overlook important aspects of parameter control [1].

– Provide a common framework for researchers to position their work and communicate their findings. Though there already exist such frameworks, we believe they are incomplete in some aspects, thus, prompting researchers to ignore important aspects of the problem or promising directions of research.

[1] As examples we mention the lack of work considering the feedback the controller receives from the EA and considerations about how the parameter values are defined. These will be made clear later on.

– Advance the outlook of the field and the understanding of the parameter control problem by reconsidering and challenging established views on certain subjects (such as the relation between tuning and control and the specificity of a control method). Hopefully, this will also translate to practical development and progress in terms of applications and results.

In this chapter, we present our theoretical framework from three[2] different perspectives that complement each other. First, we view the problem of parameter setting as a whole and discuss the position of tuning and control within parameter setting as well as the relation between control and tuning themselves, revising the previously established perception. Second, we introduce an abstract model for a parameter controller, its components and its environment, highlighting the design process and decisions necessary, and suggesting a new control mechanism classification. Third, we align the previous model to the theory of Reinforcement Learning (RL). There are two reasons for that: (i) RL is used for implementing some of the control mechanisms described later in this thesis, thus, this theory is necessary for understanding the motivation and implementation details [3] and (ii), much more important, the translation of the parameter control problem to the RL problem points out the essential matters of parameter control, supports our controller model and brings up significant issues.

2.1 Concepts and Terms

Though parameter control for evolutionary algorithms is not a particularly complicated subject, here we define the basic concepts and discuss the terms used throughout this chapter and the rest of the thesis.

The main subject of this text is the *parameters* of evolutionary algorithms: these include **all** numeric values[4] and component choices[5] found in an algorithm. Notice that many times such values or choices are hidden behind design decisions; they are nonetheless parameters, they are just fixed to a value and made unavailable to the user. Parameters can be distinguished to two categories: *numeric* (including real and integer) and *symbolic* which have no

[2]Not counting the first subsection introducing the terminology.

[3]Though it was the author's choice to use RL to implement the control mechanisms, we do not propose that RL is the only or even the best way to go when implementing parameter control algorithms.

[4]As long as they can take more than one possible values.

[5]Given that the component can be substituted by alternatives.

natural ordering. The distinction is made because the natural ordering in the former and the lack of such in the latter influences the method of control for each.

As was explained in the Introduction, the values of an EA's parameters can greatly influence its performance, thus, the process and method of *parameter setting* is an important step in the application of an EA. Generally, EAs come with *default values* which are often suggested as generally robust settings (though this generality is often challenged - including in this thesis). Parameter setting can take the form of *tuning* and of *control*. Tuning is the process of finding well performing settings by running the EA and problem at hand several times. It is often referred to as meta-optimisation. When a parameter is tuned it must remain fixed during a run. Control refers to varying the parameter values during a run (according to any method). When a control mechanism is present, parameters are said to be *varying*, otherwise they are *static*.

We use the terms *meta-parameters* when referring to the parameters of any added mechanism for control or tuning. We also use the term *hidden parameters* to refer to any parameters of a control mechanism; the parameter(s) controlled by the mechanism no longer trouble the user at the cost of the "hidden" settings of the controller itself. That hints at the concept of second order *meta-control*, i.e. a mechanism controlling the meta-parameters of the initial controller.

The type of problem solved by an EA can also be an influential factor when considering parameter setting approaches. Following Eiben and Smith [81], we distinguish problems to *repetitive* and *one-off*. A repetitive problem needs to be solved several times[6], e.g. making the monthly schedule for the staff of an airline company. On the other hand, an one-off problem is refers to an one-time design problem for which we need to find a good solution only once, e.g. optimising the design of the power network of a country.

2.2 Parameter Setting: Tuning and Control

The first perspective of our theoretical framework that we discuss here is from the viewpoint of parameter setting as a whole and the relation of tuning and control. To this end, we use as a starting point the framework for parameter tuning as described in [80]. The essence of this framework is to distinguish three layers: the application layer (that contains a problem to be solved), the algorithm layer (that contains an EA), and the design layer (that contains

[6]Of course we mean different instances of the problem.

a method to specify all details of the given EA –that is, its numeric as well as symbolic parameters).

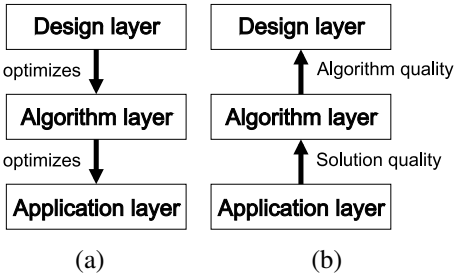


Figure 2.1: *Control flow (a) and information flow (b) through the three layers in the hierarchy of parameter tuning, after [80].*

As Figure 2.1 indicates, the whole scheme can be divided into two optimisation problems. The lower one belongs to the blocks Application + Algorithm. It consists of a problem on the application layer and an EA on the algorithm layer trying to find an optimal solution for this problem. The upper one belongs to the blocks Design + Algorithm and it consists of an algorithmic design method[7] that is trying to find optimal settings for the EA on the algorithm layer. Parameter tuning is obviously positioned in the top Design layer and the common view in the EC field has been to also position control in the same Design layer. This is in accordance with the established taxonomy introduced in [72] where parameter tuning and parameter control are two alternative approaches to setting EA parameters. According to that taxonomy, control works in an on-line fashion while tuning in an off-line fashion. Formally, the difference lies in the dependencies, or attributes, of a method. Tuning only depends on the problem at hand and the users actual definition of algorithm performance[8] while control also depends on the runtime characteristics of the EA. We could also say that a tuning method tries to solve a static optimisation problem, while a control method is concerned with non-stationary optimisation.

Though the above taxonomy captures the important difference of off-line versus on-line between tuning and control, it still views the two as (mutually exclusive) alternative approaches to the same goal. Here, we challenge this view and suggest that tuning and control serve distinctive and independent purposes and can, in fact, be combined. The rationale behind this option is a practical view of the implementation of an EA system and the temporal workflow perspective that separates the design time and the runtime of an algorithm. Adopt-

[7] As opposed to a design method relying on human intuition.
[8] This is called *utility function* in [80].

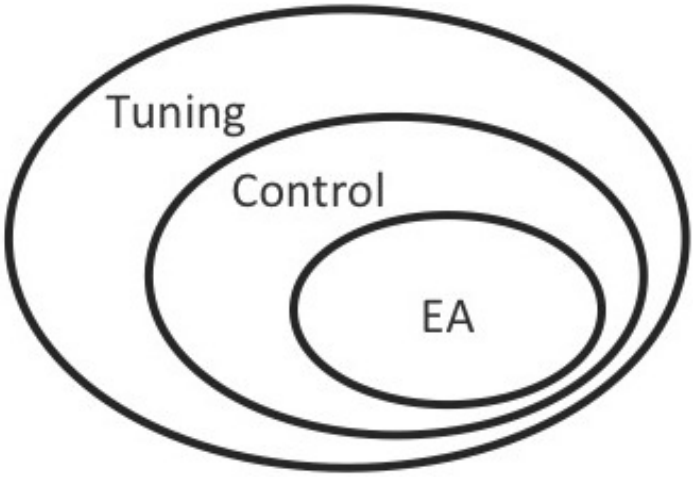


Figure 2.2: *Parameter setting as layers.*

ing this view, parameter control is a part of the deployed runtime system with the task of online variation of parameter values (the benefits of which are discussed later) while tuning is a procedure in the design phase that aims to tailor the overall system to a specific problem (class). This view also brings forth another point: a controller does have parameters of its own (though these are frequently hidden behind design decisions as mentioned earlier). Therefore, the controller can be tuned as part of the integrated search algorithm (instead of tuning the EA parameters that are subjects to this controller).

The above discussion motivates our suggestion to split the control and tuning processes into separate layers. Figure 2.2 presents the parameter setting viewpoint proposed in this thesis. The core is formed by the evolutionary algorithm, a control mechanism can be added to the EA and the resulting system (with or without the controller) can be tuned. The problem solved by the EA is not shown here because the focus is the parameter setting perspective. The benefits of the tuning layer have already been established and demonstrated (see for example [247] and [80]). The performance (either speed or solution quality) of a solver can be significantly improved if its parameters are tuned. The cost is of course the time resources required for the costly tuning process.

For parameter control, the situation is quite different. The control problem is much less systematically examined although the possible advantages of control have been identified already in the past [63, 72, 76]:

– It allows the EA to use appropriate parameter values in different stages of the search process. (E.g., search by big leaps in the beginning, fine tune the near-optimal solutions by small steps in the last stage of the search.)

– It allows the EA to adjust to the changing fitness landscapes when facing dynamic problems.

– It allows the EA to collect information about the fitness landscape during the search and use the accumulating information to improve performance in later stages, (see [63], page 5).

– Using a parameter control mechanism liberates the user from the task of choosing parameter values. (That is, it implicitly also solves the tuning problem.)

While this latter argument is frequently articulated, in practice any controller must have some parameters of its own (though these are frequently hidden behind design decisions). A counter argument against this skepticism is that the meta-parameters of the controller are much less influential than the parameters of the EA, thus, they pose a less ponderous problem. Nevertheless, this has not yet been clearly proven experimentally but, in all fairness, neither of the above points has. In fact, though control has been shown to improve performance is some cases, it is still poorly understood and, in that sense, it is still an open problem. The cost of control is also less obvious: a controller may have to perform some exploration thus may slow down the search or apply parameters harmful for the rest of the search; when tuning is also applied a controller can increase the design space of the tuning process as compared to tuning the parameters of the EA.

As was made obvious above, parameter control and parameter tuning can be used in an evolutionary problem solver independently from one another, thus, we obtain four possible combinations summarised in Figure 2.3. The combinations in the top row employ tuning and have the advantage of tailoring to a specific problem class. They incur the cost of the time and resources required for the tuning process. The combinations in the left column employ control and offer the benefits of dynamically varying parameter values. They can incur the costs of slower search or increased tuning space. Here we make an important point: it is not only the approaches that differ; we also distinguish between the controllers that can be used in the two boxes of the left column of Figure 2.3. We differentiate between *tuned* controllers (that fall in the top left corner) and *out-of-the-box* controllers that function in the lower left corner. A tuned controller must be initialised through a tuning process in order to function while an out-of-the-box controller is able, as its name implies, to work without any initial calibration. Of course, as has already been discussed, any controller must have some meta-parameters, thus it could be tuned. However, the tuned controllers'

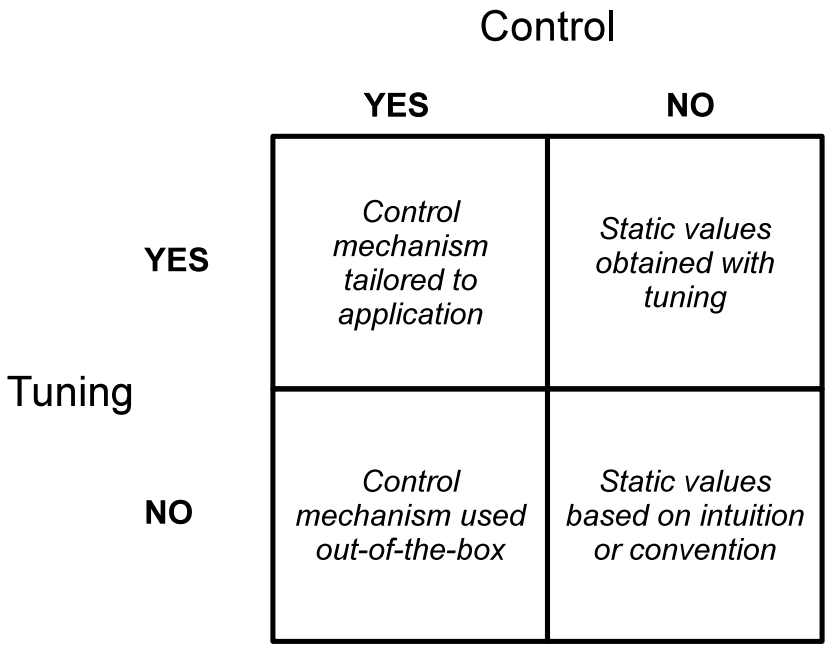


Figure 2.3: *The possible combinations of tuning and control.*

notion of calibration is extended to also include learning a good control strategy while an out-of-the-box controller is expected to be robust over any EA and problem. The difference between out-of-the-box and tuned controllers will be discussed in detail in Chapter 7. We first examine out-of-the-box controllers, so, for now, it is enough to say that such controllers function by learning control strategies on-the-fly and without any relation to the notion of tuning.

2.3 A Design Model for Parameter Control

We proceed to the second step of the framework in this chapter, which is a simple design model for parameter control mechanisms. It is a component based analysis of the working of a parameter controller, its essential parts and the manner in which they interact. This can be useful when designing a new control mechanism, especially in order to better comprehend all important design decisions and have a more systematic approach. Furthermore, it is also useful as a classification scheme for parameter controllers; as explained later, mechanisms are categorised according to three criteria: (i) the type of algorithm, (ii) the information it uses and (iii) the effects it has. These three axes of classification are similar to the "how the

changes are made", "what informs the change" and "what is changed" criteria used in [80]. However, here we take a different approach to each of these elements.

Before talking about the components and workings of the control mechanism itself, we first consider the environment it exists in, i.e. the evolutionary algorithm itself. In particular, we are interested in the *state* of an EA which determines what information the controller receives as input and is affected itself by the information that is output by the controller. Essentially, the state of an EA is described by the current population (which is what evolves over time) and, since control is our focus here, the current values of the parameters.

EA State We define the state S_{EA} of an evolutionary algorithm as:

$$S_{EA} = \{G, F, \bar{p}\} \tag{2.1}$$

where G is the set of all the genomes in the population, function $F : G \rightarrow \Re$ maps the genomes in G to fitness values and $\bar{p}$ is the vector of current parameter values.

A tuple S_{EA} uniquely specifies the state of the search process for a given evolutionary algorithm (the design and specific components of the EA need not be included in the state since they are the same during the whole run) in the sense that S_{EA} fully defines the search results so far and is the only observable factor that influences the search process from this point on (though not fully defining it, given the stochastic nature of EA operators). Time is not part of S_{EA} as it is irrelevant to the state itself; it introduces an artificial uniqueness and a property that is unrelated to the evolution. Of course, state transitions are not deterministic.

Having understood the environment, we can look at the the parameter controller itself and its composing elements. We define a parameter control mechanism as a combination of three components:

1. A choice of parameters (i.e. *what* is to be controlled).

2. A set of observables that will be the input to the control mechanism (i.e. what *evidence* is used).

3. An algorithm/technique that will map observables to parameter values (i.e. *how* the control is performed).

2.3.1 Parameters

The starting point when designing a control mechanism is the choice of parameter(s) to be controlled (as well as choices relating to when a parameter is updated). As mentioned before, we distinguish between *numeric* (e.g. population size, crossover probability) and *symbolic* (e.g. recombination operator) parameters; the natural ordering in the former and the lack of such in the latter may influence the method of control.

Notice that a control mechanism does **not** have to be specifically designed for a single parameter (though that is most often the case). On the contrary, a controller may be developed to work with any parameter or with all numeric parameters etc. Such controllers are *generic* and that approach is examined in this thesis. Nevertheless, the choice of parameters and considerations about generality and the type of parameters handled should be the starting point of the design process.

2.3.2 Observables

The observables are the values that serve as inputs to the controller's algorithm. Each observable must originate from the current state S_{EA} of the EA since, as explained above, it is the only observable factor defining how the search will proceed from this point on.

However, the raw data in the state itself are unwieldy: if we were to control based on state S_{EA} directly, that would imply that the control algorithm should be able to map every possible S_{EA} to proper parameter values. Consequently, preprocessing is necessary to derive some useful abstraction, similar to the practise of dataset preprocessing in the field of data mining. We define such an observable derivation process as the following pipeline:

$$Source \rightarrow (Digest) \rightarrow (Derivative) \rightarrow (History)$$

Parentheses denote that steps can be bypassed.

i. *Source*: As stated above, the source of all observables is the state of the EA (i.e. the set of all genomes) and the parameter values.

ii. *Digest*: A function $D(S_{EA}) = v$ that maps an EA state to a value, e.g. the best fitness found in the current population or the current population's diversity.

iii. *Derivative*: Instead of using directly a value v we might be more interested in its speed or acceleration (e.g. to make the observable independent to the absolute values of v

or to determine the effect of the previous update as the change observed in the most recent cycle).

iv. *History*: The last step in defining an observable is maintaining a history of size W of the value received from the previous step. This step includes a decision on the sliding window size W and the definition of a function $F_H(v_1, v_2, ..., v_W)$[9] that, given the last W values, provides a final value or vector (e.g. the minimum value, the maximum increase between two consecutive steps, the whole history as is etc.).

The amount of information in the source S_{EA} and the multitude of options for the derivation steps described above shows that one could define an endless number of observables. However, up to date only a handful of options have been used (mainly fitness based and, in some cases, diversity) and there has been no in-depth investigation regarding their influence and whether their choice depends on the EA or the controller.

2.3.3 Algorithm

The core of the parameter controller is the algorithm that maps a vector of observable values to a vector of parameter values. Any technique capable of such a mapping can be used as an algorithm for the control mechanism, e.g. a rule set, an ANN or a function are all valid candidates. The choice of the proper algorithm seems to bear some resemblance to choosing an appropriate machine learning technique given a specific task or dataset. Whether EA observables display specific characteristics that make certain biases and representations more suitable is a question that needs to be investigated[10]. In any case, it is obvious that given the type of parameter controlled (i.e. numeric or nominal) different techniques may be applicable or not.

Regardless the algorithm and representation used, we distinguish between two main categories of control techniques, based on a fundamental characteristic of the controller: whether it is static or it adapts itself to the evolutionary process.

i. *Static*: A static controller remains fixed during the run, i.e. given the same observables input, it will always produce the same parameter values output. In other words, the

[9] Notice that indices do not relate to absolute time but indicate an ordered sequence of W elements

[10] As well as the reverse question, i.e. whether certain observables will be more suitable given a specific algorithm.

values produced only depend on the current observables input:

$$\vec{p} = c(\vec{o}) \;\; and \;\; \vec{o_1} = \vec{o_2} \Rightarrow c(\vec{o_1}) = c(\vec{o_2})$$

where $\vec{o} \in O$, $\vec{p} \in P$ are the vectors of observables and parameter values respectively and $c : O \mapsto P$ is the mapping of the controller.

ii. *Dynamic*: A dynamic controller changes during the run, i.e. the same observables input can produce different parameter values output at different times. This implies that the controller contains an internal state. The parameter values output depend on both the current observables input and the controller's current state; the controller's internal state also changes according to the observables input and the current state:

$$\vec{p_t} = c_p(\vec{o_t}, S_C^t) \;\; and \;\; S_C^{t+1} = c_S(\vec{o_t}, S_C^t)$$

where $\vec{o} \in O$, $\vec{p} \in P$ are the vectors of observables and parameter values respectively, $S_C \in \mathcal{S}$ is the state of the controller, $c_p : O \times \mathcal{S} \mapsto P$ is the mapping of the controller and $c_S : O \times \mathcal{S} \mapsto \mathcal{S}$ is the controller's state transition function.

According to this classification, a time-scheduled mechanism is a trivial case of a dynamic controller; it maintains a changing state (a simple counter) but is "blind" to the evolutionary process since it does not use any observables. Furthermore, note that this model also covers control mechanisms that are not necessarily separate and distinct components, e.g. self-adaptation in ES can be seen as a dynamic controller that implicitly maintains a state influenced by the evolutionary process.

2.4 Parameter Control as Reinforcement Learning

The last step of the theoretical framework and this chapter is from the perspective of Reinforcement Learning (RL). We expand on the decomposition of the parameter control mechanism presented in the previous section, discuss the analogies with a Markov Decision Process (MDP) and formulate EA parameter control as a reinforcement learning problem. We do not suggest that using RL for implementing parameter control is necessary; nonetheless, we believe that this formulation is a valuable viewpoint of parameter control that gives us the tools to comprehend important aspects of the matter, especially the relation of the observables to the dynamic (or not) nature of the environment, the importance of how parameters

are changed and the complexities of defining a reward signal. From a practical point of view, mapping parameter control to RL gives us the benefit of a wealth of existing research and allows us to employ existing algorithms that have been refined over the course of several years. This section requires an understanding of the basic concepts of reinforcement learning (a short introduction to RL is given in Appendix A).

As was explained in the previous section, the complete state of an EA can be defined as $S_{EA} = \{G, F, \bar{p}\}$ and the definition of an EA parameter control method requires three components: (i) the choice of parameters (including the frequency and time of updates), (ii) the observables that are derived from the search state S_{EA} and are used as input to the controller and (iii) the specific mechanism/algorithm (either static or dynamic) that maps the current observables' values to parameter values. The state of the EA, and in extension the outcome of the search process, can be affected by changing G or $\bar{p}$. Evolutionary operators change G; every time variation or selection is applied, the genotypes in the population change. Controlling the parameters changes $\bar{p}$, thus also affecting the next state(s) of the EA. The goal of such control is to maximise the final performance (at the end of a number of available evaluations). However, the long term effects of a certain $\bar{p}$ applied cannot be known, only the resulting S_{EA}.

There is an apparent analogy between the EA parameter control problem and the full RL problem and a Markov decision process. The EA state observables suggest a state space, the parameters controlled and their ranges point to an action space while a dynamic control mechanism component could be implemented by a specific RL algorithm. Although this high level mapping is straightforward, the exact formulation of the RL problem is far from trivial. To define an MDP we need to define a state space, an action space, a transition function and a reward function [263] [276]. These are explained below.

2.4.1 State and Transitions

The most important and challenging is the formulation of the state space. Ideally, we would like our state definition to give our process the Markov property, i.e.:

$$P(s_{t+1}, r_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}, r_{t+1}|s_t, a_t) \tag{2.2}$$

This means that the next state depends only on the current one and is independent of the sequence of states before that. In other words, a state definition would entirely contain

all the necessary information for deciding the most appropriate action without the need of knowing the complete history that led to the current state [276].

Here we would like to distinguish between what we *define* as state and what we *observe*. Though it is only a conceptual distinction (since we're practically only concerned what is actually observed), it does influence the way the problem is seen and approached. For both the MDP's defined state and the observations made by the EA, the options are to use directly the S_{EA} or derived observables. The possible combinations are discussed below and summarised in Table 2.1.

First, using directly S_{EA} (i.e. both to define the MDP state and to observe) is problematic. It is unlikely such a state will be repeated, thus whatever is learned there is useless, while approximations of a state value function seem rather difficult. Obviously, the observations made must be based on derived observables. If we keep S_{EA} as the defined state, then we are considering a Partially Observable MDP (PO-MDP) setting because multiple S_{EA} states can give the same observable values [257]. Approaching the problem as a PO-MDP is challenging in terms of algorithms capable of tackling it. The alternative would be to define the state space using derived observables as its dimensions. The likely risk in this case is that the Markov property is lost and the problem becomes non-stationary. If the observables used are too simple or weak then not only the current state will not contain enough information but also the optimal actions for a state will quickly change. Also, when using the observables to define state, the type of observables may also be influential. History observables may favour the Markov property (since they include information about previous states by definition) but is not clear if non-history observables could also sufficiently reflect the S_{EA} and maintain the Markov property.

Furthermore, regardless the approach, observables will likely be continuous, so will the state space of the controller. Prior discretisation of the state space can be highly inefficient since the distribution of values of the observables cannot be known beforehand. Consequently, special techniques are required to deal with the continuous state space [273].

Table 2.1: *MDP state definition and EA observations combinations with the corresponding issues.*

MDP State	**Controller Observed State**	**Concern**
S_{EA}	S_{EA}	Impractical
S_{EA}	Derived Observables	PO-MDP
Derived Observables	Derived Observables	Non-stationary

Regardless the exact definition of the state space, the transition function is unknown

because of the unknown effect of parameter values and because of noise due to the influence of the stochastic operators on the state of the EA.

2.4.2 Actions

Second, we must define the actions of the RL problem that will give parameter values. There are two general options for approaching this: (i) actions that increase or decrease (or maintain) the current parameter value and (ii) actions that set a specific value to the parameter. These options pose a tradeoff. Defining actions to increase/decrease the current parameter values makes the next values dependent on the current ones. Subsequently, we should expand the state definition to include the current parameter vector or we should cope with another source of uncertainty about the current state making the PO-MDP problem described earlier stronger. On the other hand, actions that define direct values for parameters mean that the action space must be the Cartesian product of the domains of all parameters controlled. Such an action space also poses the problem of continuity since actions correspond to values of continuous parameters. However, unlike the state space, discretisation of the action space may be reasonable. Though we can expect that certain regions of parameter values may deserve more attention and finer granularity, some parameter have wide "good areas" of values (e.g. crossover rate of GAs). Even so, a reasonable amount of discretisation intervals would be required resulting in a higher number of total actions than if we used option (i), thus slowing down learning because more exploration would be required.

2.4.3 Reward

Finally, we define reward. According to Sutton and Barto [263] reward must indicate what we want to accomplish without being biased by the designer's intuition about the "how".

Depending on the target application of the controller (Figure 2.3) different options may apply. If a controller must be calibrated off-line, then the final best fitness of an evolutionary run is a straightforward option for the reward. In that case, we have a Monte Carlo approach where multiple training runs are made with the final "score" of each taken into account.

For an off-the-shelf controller the situation is different. Such a controller operates in a single continuing task (a single evolutionary run). Consequently, we require a reward that is calculated after each parameter change (e.g. iteration of the EA) from the beginning and throughout the run but, at the same time, is able to approximate the goal of an optimal so-

lution achieved at the end of the run. Some options could be the difference between the previous and the current best fitness or a binary signal determining whether any improvement took place since the last action. We cannot directly determine what would be the best solution; the issue of defining reward for an off-the-shelf controller will be the subject of further study and experimentation in this thesis.

No matter the exact definition, a model of the reward function is not available because of the unknown effect of parameter values and the stochastic nature of the evolutionary operators.

3

A Survey of the Field

Now that the basic theoretic framework and concepts have been explained, we proceed to a detailed survey of existing work in the field of parameter control for evolutionary algorithms. As the reader will see from the length of this chapter, the number of works reviewed and their time span, the field of parameter control is neither new nor unpopular. The concept of (self-adaptive) parameters appeared shortly after the algorithms themselves and, since then, a multitude of works has been published on the subject. However, we will show that there is a lack of focus, continuity and generality in this work - with a few exceptions - which translates to a lack of adoption in the real world and practical applications. Consequently, we believe that the field can benefit from a new impetus and a more comprehensive and practical approach. These points will be made clear through the review and clustering of relevant work and the trends and challenges that will be identified in these.

Thus, the aim of this chapter is to:

i. Discuss the map of the field from a 'helicopter view'.

ii. Provide an extensive review of related work.

iii. Observe main clusters of work and analyse the corresponding research trends.

iv. Identify the most prominent challenges for future developments.

The next section gives a short overview of the development of the field followed by a section with an overview of control mechanisms for each of the major parameters / components of an EA. The next two sections are devoted to approaches that control multiple parameters and to parameter independent methods. The chapter is concluded with an overview that identifies the most important trends over the last two decades and discusses future directions.

3.1 Development of the Field

The problem of parameter control in EAs is as old as EAs themselves. However, it was not noticed in the early years of the field for a number of reasons. One of those reasons was the general attitude that perceived and advocated genetic algorithms (GAs) –that were the best known EAs back in the 70's and 80's– as robust search methods that work well for a wide range of parameter values. In other words, even the tuning problem was not recognised widely, although some authors discussed 'optimal' values for population size, mutation rate, and crossover rate, see e.g. [62, 116, 241]. Concerning the runtime adaptation of parameter values the research community was divided. Among GA researchers the issue remained largely unnoticed, with just a few exceptions [242], and the same was true for Evolutionary Programming. Meanwhile, the use of self-adaptation mechanisms to change mutation step sizes on-the-fly was standard practice in Evolution Strategies [226, 244].

Another problem hindering development was the fragmentation and inconsistency in terminology as illustrated by the notion of self-adaptation. The term was introduced in Evolution Strategies to denote the technique of including parameters of the EA (in particular, the mutation step sizes) into the chromosomes, thus co-evolving solutions and strategy parameters. However, some authors used a different name for the same technique, e.g. [98] used the term "meta-evolution" for self-adaptation, and some others used the same name for a different technique, e.g.[77] used the term self-adaptation to denote an adaptation mechanism that did not encode parameter values into the chromosomes. By the end of the 90's the idea of changing parameter values on-the-fly gained traction even in the GA community, [252, 253, 269], but there was no unifying vision and generally adopted terminology. Some authors recognised the importance of these issues and offered a (partial) solution, see [11, 251], but these attempts did not receive the attention they deserved.

The situation changed in 1999 with the publication of [72] by Eiben et al.. This paper presented a unifying vision, a clear taxonomy by a list of essential features and the corresponding terminology that were quickly adopted and became the *de facto* standard in the field. It categorised parameter setting methods according to four axes: (i) which parameter is changed, (ii) the type of control, (iii) the scope of change and (iv) the type of evidence used. It clearly defined the three types of control, which are almost always used in parameter control texts: *deterministic*, *adaptive* and *self-adaptive* (as were explained in Chapter 2). The other axes of the taxonomy, however, did not have the same impact and are often overlooked in the field. Not rightly so. Here we consider the type of evidence/observables to play an important role in parameter control as explained in Chapter 2; their treatment, though, in [72] was too limited and did not bring up the essential points of the matter. Furthermore, though the first axis of which parameter is changed is (unavoidably) recognised when discussing a parameter controller, the relevant issues of applicability, generality and the patchwork problem (which will be made clear by the end of this chapter) are rarely considered.

Apart from [72], several other frameworks and views of the field have been discussed. Here, we provide a list of relevant PhD Theses in this area, paying special attention to their contributions to classifying the field by alternative or complementary taxonomies.

– J.E. Smith, Self Adaptation in Evolutionary Algorithms, 1998 [251] presents its own taxonomy that is very similar to the one we presented above. However, the evidence that guides the changes is seen as "perhaps the most important" dimension (page 18). Later on in a joint publication, these views were merged [81] (page 142).

– R.K. Ursem, Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization, 2003 [270] contains a discussion of methods for parameter control in EAs and a critical view on existing taxonomies therein. It proposes a novel taxonomy (page 50) that basically distinguishes "non-adaptive control" and "adaptive control", where the first one lumps together parameter tuning and deterministic control in the scheme of [72], while the second one is further divided into "measure-based", "self-adaptive", and "population-structure-based".

– O. Kramer, Self-adaptive heuristics for evolutionary computation, 2008 [167, 168] is primarily concerned with self-adaptive forms of parameter control. However, it also contains an "Extended Taxonomy of Parameter Setting Techniques" (page 31) that

indeed extends the taxonomy of [72]. It does maintain the 3 types of control methods (deterministic, adaptive, self-adaptive) and identifies three types of parameter tuning: "by hand", by "Design of Experiments", and by "metaevolutionary" techniques.

– J. Maturana, General Control of Parameters for Evolutionary Algorithms, 2009 [196] (in French)[1] presents a good model of parameter control that distinguishes the EA, the controller, and their interactions, thus allowing for systematic developments (page 44).

– A. Fialho, Adaptive Operator Selection for Optimization, 2010 [93] does not introduce a new classification scheme, but uses the terms "off-line" or "external" tuning (page 35) and "on-line" or "internal" parameter control (page 38). This terminology emphasises the conceptual similarities with the reactive search perspective where off-line tuning and on-line tuning coincide with our parameter tuning and adaptive parameter control, respectively, cf. page 159 in [28].

– A. Aleti, An Adaptive Approach to Controlling Parameters of Evolutionary Algorithms, 2012 [8] focuses on adaptive parameter control and presents an interesting model (page 57) that distinguishes the optimization process and the control process. This latter is further divided into four principal steps or strategies for: (i) Feedback Collection, (ii) Parameter Effect Assessment, (iii) Parameter Quality Attribution, (iv) Parameter Value Selection. These identify the essential components and provide a natural division of research, i.e., a sub-taxonomy within the adaptive branch of [72].

The theses listed above cover a very substantial amount of work related to parameter control and provide valuable insights about the most specialized experts' view on the field. One interesting idea is to shift the main 'water shed' from tuning vs. control to blind (tuning and deterministic) vs. informed (adaptive and self-adaptive) – rephrased after [270]. The other notable issue is the possibility to distinguish fine grade details within the adaptive parameter control category as offered in [251] and [8]. These details concern the fourth dimension above regarding the evidence that guides the changes and the way it is collected and handled.

[1] see http://www.inf.uach.cl/maturana/files/presentation_WEB.pdf for a good English summary in the form of a presentation

3.2 Parameter specific methods

In this section we present parameter specific strategies that were designed for a certain parameter or component. Though some may be considered applicable to other parameters as well, they are placed in the section dedicated to the parameter they were originally intended for in literature. The following subsections describe control designed for the population, variation, selection, fitness function and for parameters related to parallel EAs.

3.2.1 Population

Extensive work has been carried out regarding the population size of evolutionary algorithms. Here we present this work divided in four main categories:

i. Theoretical studies on population size and the benefit of dynamic population sizing.

ii. Approaches that aim at disposing of the population size parameter all together by introducing a new operator or concept.

iii. Strategies that attempt to approximate a good (minimal and sufficient) population size during the EA run.

iv. Mechanisms that directly control the parameter on-the-fly.

A literature review on control of the population size is given by Lobo and Lima [188].

Theoretical studies

The population size as a parameter has been studied from a theoretical point of view within the Genetic Algorithms community employing schema and building block theory. There are two ways to approach the population size: (i) how large should the population be to have a sufficient supply of building blocks and (ii) how large should the population be to cope with errors in selection [127].

Reeves [227] calculated the minimum necessary population so that every allele is present for binary as well as n-ary representations. It is suggested that the population is initialized in a "smart" way so as to cover as much of the search space as possible. Though having all alleles present in the initial population is important, it is more crucial that actual building blocks are also available. Goldberg et al. [110] derive the probability that all building blocks are present in a population and, based on that, an equation that calculates a minimum

necessary population size as a function of the alphabet cardinality, the size and the number of building blocks.

From the selection point of view, Goldberg et al. [108] used statistical decision theory to derive a formula for the necessary population size based on the permissible decision errors between building blocks, the noise caused by selection and variation and the complexity of the problem. Harik et al. [127] took an alternative approach to modeling population size requirements according to selection errors. They use the theory of random walks and the Gambler's Ruin model to approximate selection decisions of a GA with tournament selection and without mutation.

More recently, Laredo et al. [175] proposed a method for evaluating GA population control schemes, based on building block concepts and decomposable functions with known population dynamics [108]. The authors examine the assumption that larger populations are useful in the beginning of the run, while later on, the EA can converge with smaller population sizes. Their approach is theoretical and limited only to selectorecombinatorial[2] GAs and decomposable trap functions.

A similar theoretical study on dynamic population sizing for GAs is given by Lobo [186]. Population size is examined from the perspective of the building block concept [108] and the Gambler's Ruin model for population size [127]. The latter calculates the minimum requirement for population to solve a problem with a given probability. It uses the supply of building blocks and the probability that selection will correctly favor a building block instead of a competitor. The author derives two control strategies based on the observation that these factors of supply and selection do not remain constant during the run of the GA. The building block supply model provides an approximation method for the former while the latter is calculated using a heuristic. These values are used in each generation with the Gambler's Ruin model to derive the new population size.

Removing the population size as a parameter

The approaches described here remove the population size parameter mainly by replacing it with a maximum lifetime, by imposing a limit on some resource that indirectly limits the population or by replacing the multi-individual population with a representative distribution.

Removing the population size as a parameter firstly appeared with the Genetic Algorithm with Varying Population Size (GAVaPS) by Arabas et al. [12]. Individuals are assigned a

[2] A selectorecombinatorial GA uses only selection and recombination, i.e. it is a GA without mutation.

maximum lifetime which depends on their fitness and the average or best fitness of the population. When this maximum lifetime is reached, the individual is removed. The ratio of offspring to population size is kept constant, thus selective pressure also remains fixed while the population size varies. Deciding the maximum lifetime of an individual requires minimum and maximum bounds, consequently, two new parameters are introduced.

The lifetime concept of GAVaPS was extended using non-random mating: niGAVaPS by Fernandes et al. [88] uses an incest prevention mechanism (inbreeding control) , while nAMGAVaPS by Fernandes and Rosa [86] selects mating partners based on phenotypic similarity (assortative mating control). SRP-EA [87] further extended nAMGAVaPS by introducing (i) a dynamic threshold of similarity for the assortative mating mechanism and (ii) a probabilistic scheme to individual removal instead of the fully predetermined lifetimes in the original GAVaPS and nAMGAVaPS.

Another variation of the GAVaPS lifetime scheme was described by Bäck et al. [22]. The Adaptive Population GA (APGA) uses the same lifetime allocation but differs from GAVaPS in the reproduction cycle and in that, when incrementing the ages of individuals in each cycle, the best individual of that cycle remains unchanged. This adaptive population strategy is part of an ensemble and is described in more detail in Section 3.3. This method was also applied to co-operative co-evolution by Iorio and Li [147]. An analysis of APGA by Lobo and Lima [187] shows theoretical and experimental results suggesting an upper bound and a converging population to a constant size that is determined by the minimum and maximum lifetime parameters. The authors conclude that the population is not adapted by the APGA but the size parameter is in fact replaced by the two lifetime parameters.

Cook and Tauritz [55] suggested two strategies for removing the population size parameter. FiScIS-EA removes individuals according to a survival probability, derived by linear scaling of the individual's fitness in the range between the minimum and maximum fitness values present in the population. GC-EA simply evades choosing a population size by maintaining a population as large as is allowed by memory limitations. This requires a well-chosen parent selection mechanism that scales well while a survival selection operator is still needed in case the memory boundary is reached during the run. Both methods aim at resolving the issue of the population size without introducing new meta-parameters.

Another method for removing the population size parameter that is particular to Genetic Programming was suggested by Wagner and Michalewicz [279, 280]. The population size and maximum tree depth parameters are replaced by two parameters limiting the maximum total number of nodes in the population (soft and hard limits) in order to bound resource

consumption. As a result, the number of individuals in the population varies with time, allowing the natural growth of good quality complex solutions with the expense of having less individuals. This approach was used in the DyFor algorithm [281] that also controls additional parameters and is thus described in Section 3.3.

Finally, we also mention here two genetic algorithms that replace the population with a self-adapting probability distribution. The Population-based Incremental Learning algorithm by Baluja and Caruana [26] maintains a probability vector to represent its population. Each component of this vector represents the proportion of alleles $0/1$ in the assumed population. A similar approach is the Compact Genetic Algorithm by Harik et al. [129].

Approximating a good population size

The following methods automate the process often performed by human designers: running the EA with progressively larger population sizes to determine how large a population would be sufficient for solving the problem at hand. They differ mainly in the number of subpopulations they use and the criteria for terminating subpopulations.

Harik and Lobo [128] first suggested the Parameter-less GA.[3] It creates and runs separate populations, each new population is double the size of the previous. Populations are run in parallel and raced: when a population is outperformed by a larger one, it is immediately deleted. Smaller populations are given more evaluations than larger ones. The target is to find as soon as possible the smallest population capable of solving the problem.

An adaptation of the Parameter-less GA above is the Greedy Population Sizing approach (GPS-EA) by Smorodkina and Tauritz [256]. New subpopulations again have double the size of the previous. The difference is that GPS-EA runs only two populations in parallel at any time. The smaller population expires either if its average fitness becomes worse than that of the larger or if its best fitness stops improving at a value better than that reached by its predecessor. When a population expires, the algorithm proceeds by creating a new one. A rough theoretical analysis suggests that the GPS-EA should be able to reach fitness levels comparable to that of a GA with an optimal static population in no more than double the time.

The IPOP-CMA-ES by Auger and Hansen [15] and the IPOP-αCMA-ES by Hansen and Ros [126] extend the (μ_w, λ)-CMA-ES [122]. The population control simply augments the

[3] The authors use the name "Parameter-less" for their algorithm. However, this name is somewhat misleading, since technically speaking their algorithm is not parameterless and only concerns population sizes.

existing restart strategy of the algorithm by doubling the population size with each restart.[4] A further extension is suggested with the BIPOP-CMA-ES by Hansen [121, 120]. With each restart, two interlaced methods for calculating two options for the new population size are used. A large population size is doubled every time while a small population size is proportional to the ratio of the last used large size to the default size modified by a random factor. The new population size is the smallest of the two options.

Controlling population size on-the-fly

Contrary to other parameters, control of the population size entails some additional design choices except for the standard parameter control components (frequency, amount of change etc.). The additional components are the procedures by which individuals are created or removed when the population grows or shrinks. Costa et. al. [56] investigate the potential of a simplistic deterministic control schedule of monotonous growth or shrinking. Different combinations of creation/removal strategies are tried, however, the presented experiments are too limited to draw meaningful conclusions.

Deterministic control can of course be more elaborate than a monotonous change. The saw-tooth GA by Koumousis and Katsaras [165] combines a linear decrease schedule with periodic reinitializations that restore the population to its maximum size (by inserting random new individuals). An inverse saw-tooth model of the population size is examined by Hu et al. [138]. Inspired by a parallel and asynchronous EA implementation, it gradually increases the population while mass extinctions occur periodically. Experiments suggest that these extinctions can trigger increases in the best fitness. De Vega et al. [66] also used extinctions that are triggered in every generation and remove a fixed number of the worst individuals.

The simplest form of adaptive population size is based on intuitive triggers or formulas. PRoFIGA by Eiben et al. [75] observes the improvement of the best fitness to make adjustments: increasing fitness triggers a proportional growth of the population, stagnation for a period longer than a predetermined threshold results in a fixed increase while in all other cases the population diminishes. The aim is to create appropriate exploration-exploitation levels in the beginning of the search, during the hill-climbing process and after the population is stuck at a (local) optimum respectively. Fernandez et al.[92] suggested using extinc-

[4] The text refers to the parameter controlled as "population size λ" which might be confusing since λ usually notes the number of offspring produced in each generation. This is due to the structure of the CMA-ES, which differs from the conventional EA loop, and also the fact that it is defined $\mu = \frac{\lambda}{2}$.

tions that are triggered by events/metrics defined as policies to be chosen by the algorithm designer.

Smith and Smuda [254] proposed dynamic adaptation of the population size of GAs based on a theoretical foundation rather than intuition. Using theory on schemata fitness, schemata competition and selection error [109], the adaptive control mechanism continuously estimates the expected fitness loss due to selection error by performing pairwise competitions of the common schemata of mating pairs and subsequently assigns the number of offspring to each mating pair so as to maintain a loss close to user defined target value. Though one parameter is replaced with another, the motivation of the authors is to replace obscure GA parameters with more intuitive and user-friendly ones.

Using a clustering algorithm, a GA can construct a linkage model and derive building blocks information (such as the DSMGA [296]). Yu et al. [297, 298] used this information, e.g. fitness variance of building blocks and signal size between competing building blocks, to estimate the necessary population size for the next generation. They suggested performing population growth with random individuals in the beginning and using crossover in the final stages.

Hinterding et al. [134] combined adaptation using fitness as feedback and testing multiple subpopulations of different sizes. The proposed scheme maintains three populations of different sizes. A run is divided in epochs and, at the end of each epoch, population sizes are changed according to a set of simple rules that move the population sizes towards the size that performed best in the last epoch or expand the range of used sizes if the subpopulations performed equally. Unlike the "optimal size approximating" methods of the previous subsection, this control strategy continuously adapts the population size based on feedback from the search and can both increase and decrease it. This strategy is combined with self-adaptive mutation in the Self-Adaptive Genetic Algorithm described in 3.3.

Finally, population size has also been self-adapted. Since it is a global parameter, its value has to be derived from the individuals' values. Eiben et al. [79] calculated the population size of the next generation as the sum of the values of all individuals. Teo [264] derived the population size by averaging the individuals' values and also suggested encoding the rate of change for the parameter value in the genome instead of absolute values. We mention that the former study concluded that self-adaptation of the population size is not promising while the second had encouraging results. However, they differ in many aspects, most notably the EA variants used.

3.2.2 Variation

Controlling the various aspects of the variation operators (i.e. in general crossover and mutation in all their namings, forms and flavors) is probably the most popular subject and focus point found in literature on parameter control in EAs. In this section we present work carried out in the following main categories:

i. Theoretical studies on the influence of variation operators' parameters and attempts to determine theoretic optimal values for these parameters.

ii. Removing variation parameters by using novel mutation and crossover operators.

iii. Control of mutation and crossover rate (commonly noted as p_m and p_c respectively - or F and CR in Differential Evolution).

iv. (Self-)Adaptive Operator Selection (AOS). Unlike (3), studies in this category control the selection of available operators including several choices, treating them as alternatives (e.g. choosing among several types of crossover).

v. Control of the distribution of offspring sampling, a subject that is particular only to real number encodings but has been the focus of extensive work.

3.2.2.1 Theoretical studies

Theoretical results on the values and effect of the mutation probability in Genetic Algorithms were given by Hesser and Männer [132]. They suggest that mutation is only necessary for finite populations to compensate with the loss of building blocks due to errors of the selection process. In the absence of that risk, i.e. in the hypothetical situation of infinite population, mutation would act disruptively by destroying already accumulated information and, thus, reducing the convergence speed. Furthermore, it is suggested that the probability of mutation should converge to 0 as the genome length l increases because the probability to destroy good value combinations increases proportionally with l. Using a time-requirement model and based on the assumption of a sufficiently large population and certain absorption probabilities, a formula is derived for calculating optimal mutation rates. A further heuristic for calculating a good combination for mutation and crossover rates is proposed but it requires estimating values that are very difficult to derive. Böttcher et al. [34] presented another analysis limited to a $(1+1)$-GA and the LeadingOnes problem showing that

the optimal mutation rate is not the conventional $\frac{1}{n}$. They suggest an equation for adaptive mutation rate that reduces the expected time to solution .

Jansen and De Jong [148] presented a study on the influence of the number of offspring λ on the number of evaluations required to reach the optimum for a $(1 + \lambda)$-EA. Analysis with two simple functions (OneMax and LeadingOnes) suggests that a $(1 + \lambda)$-EA cannot outperform a $(1+1)$-EA when solving unimodal functions, though keeping λ within certain bounds (proportional to $\log N$ - where N is the population size) will not incur a significant slowdown. Further analysis with artificial multimodal landscapes shows that a value of λ larger than one can greatly increase performance when solving multimodal problems. Since keeping within the bound of $\log N$ does not slow evolution significantly while larger λ values are beneficial for multimodal landscapes, the authors conclude with the generic guideline of defining $\lambda = \log N$ when the characteristics of the fitness function are not known.

In the field of GAs, crossover is often seen as the main search operator while mutation is considered as a secondary operator [136], useful only for the recovery of lost allele values [132]. Subsequently, traditional practice often involves setting mutation rate to very small values [62]. Mühlenbein [211] discussed a different view: mutation is itself a search operator, especially beneficial when combined with high selective pressure [19]. This latter line of thought suggests that mutation rate should be set to relatively high values. Bäck [18] used mutation as the only search operator and studied the success probability and optimal mutation rate for a simplistic problem type.

De Jong and Spears [64] examined the disruptiveness of crossover in Genetic Algorithms and its dependence on the number of crossover points. They suggested that crossover disruption is beneficial late in the search when the population is converging and when the population size is too small and that an appropriate adaptive crossover operator should increase its disruptive potential when population homogeneity increases.

In the area of Differential Evolution (DE), it was originally suggested that DE is quite robust and that its variation parameters (scaling factor and crossover rate) can be easily set to some "standard" values [262]. However, like with the other variants of EAs, later studies showed that the performance of DE is also highly influenced by the values of its parameters [305], [38], [104]. Reynoso-Meza et al. [228] conducted experiments to find good settings for these parameters when DE is applied to a multi-objective problem class and concluded that the choice for the value of the scaling factor is difficult and case specific.

3.2.2.2 Removing variation parameters

Fernandes et al. [91] introduced a novel mutation operator based on the nature-inspired concept of self-organized criticality and the sandpile model (for more information see [25, 24]). At each generation, individuals are mapped to a matrix (each row being an individual and each column a gene). This matrix is used as the lattice for a sandpile model; whenever an avalanche occurs, the affected cells (i.e. genes) are mutated. To avoid mutating highly fit individuals, an avalanche is interrupted at a cell if a randomly generated value is higher than the normalized fitness of the individual that the specific cell belongs to. Since the sandpile mutation must work on evaluated individuals and the subsequent selection must have up-to-date (i.e. after mutation) fitness values, that would require two cycles of evaluations per generation. To avoid this, the sandpile mutation uses the fitness values of the parents of an individual to derive an expected normalized fitness, thus requiring evaluations of individuals only after the mutation procedure is complete [89, 90].

3.2.2.3 Controlling mutation and crossover rates

An adaptive approach to controlling both mutation and crossover rates was presented by Srinivas and Patnaik [260]. The goal is to maintain diversity by increasing variation when the population converges (which is detected by the mean fitness approaching the best) while also maintaining good solutions. To achieve this, every time an offspring is produced, crossover rates are defined by a linear function proportional to the difference in parent's fitness with the best and inversely proportional to the grade of convergence. A schema analysis suggests that this adaptive method is better than a static approach in terms of promoting schemata with higher fitness and rapidly increasing the fitness of schemata.

Bäck [19] applied self-adaptation to the GA mutation rate parameter. Bit string genotypes are extended with one or multiple sections encoding mutation rates. These sections are decoded into probabilities and used to mutate themselves; the resulting new mutation rates are then used to mutate the objective values. This approach differs from the original ES self-adaptation where mutation parameters are not self-mutated but instead a static distribution is used. In following work [23] the same author suggested that the binary encoding of the mutation rate limits precision and fine-tuning. As an improvement, mutation rate is attached as a real number to the genotype. Furthermore, self-mutation of p_m is performed using a lognormal distribution similar to the common practice in Evolution Strategies. In a different direction, Smith and Fogarty [253] applied GA self-adaptation to a steady state GA

and investigated the influence of selection and crossover on the success of self-adaptation.

More recently, Vafaee [272] and Nelson suggested transforming the mutation procedure of a GA into a sampling process in the $\{0,1\}^n$ space and, subsequently, controlling this distribution by adapting a vector of probabilities . The Site-Specific Rate GA (SSRGA) calculates a probability vector using a fitness weighted sum of a set of individuals. The population is split into several subpopulations that occupy different peaks and the probability vector for each subpopulation is calculated. Individuals are then mutated by setting their alleles according to the probabilities of the corresponding vector. Though it is described as mutation by the authors it is in fact a sampling process that resembles the method of the CMA-ES [119] (see Section 3.2.2.5). A similar method was presented by Nadi and Khader [213] with the difference that the sampling process is merged with crossover: two parents create two offspring; alleles that are common to both parent are maintained while the rest are set according to a probability vector derived from the whole population.

Extensive work has been carried out for the control of the scaling factor and crossover rate of Differential Evolution (DE). The simplest methods are based on formulas or rules according to the authors' intuition of how the parameters should vary. Ghosh et al. [105] [106] suggested a strategy where F decreases (linearly or logarithmically) as the distance of the target vector's fitness to the current best fitness decreases while CR increases as the fitness of the donor vector is closer to the current best fitness. A formula specific to multi-objective optimization for adapting F using the numbers of non dominated solutions and crowding metrics was proposed by Qian and Li [224]. Liu and Lampinen [184] used fuzzy logic controllers with fitness and diversity measures as inputs for the FADE algorithm. The fuzzy memberships are used to evaluate rules that are hand coded by the authors.

For a more adaptive approach, Zielinski and Laur [302], [303] used the Design of Experiments method: for the two variation parameters of DE, a two-by-two factorial design is used and all combinations are applied for equal numbers of trial vectors; when significant differences are detected the two points of the affected parameter(s) are moved in the appropriate direction by a predetermined interval. A popular approach for the adaptive control of the DE variation parameters is sampling their values from random distributions and adapting the characteristics of these distributions. The JADE algorithm by Zhang and Sanderson [301], [300] samples the values for the two parameters from two corresponding normal distributions for each new trial vector created. The values that lead to a vector better than the parent are saved and used to calculate the means of the distributions for the next generation. A similar approach is followed by the ILSDEMO [290] and SaDE [225] algorithms (which

are ensembles and are described in Section 3.3).

Self-adaptation of the DE variation parameters has also been explored by multiple authors, the differences lying in the manner the offspring's parameter values are calculated. Abbass [1] used the typical mutation formula of DE, Brest et al. [38] gave the trial vector the value of the target vector with 90% chance (otherwise random) while the SA-DE algorithm of Brest et al. [40] averages the parameter values of the target vector and the three differential vectors and mutates this average with a factor from a lognormal distribution (following ES practice).

3.2.2.4 Operator selection

Adaptive Operator Selection (AOS) deals with the symbolic parameter of the variation operators. Several alternatives for such operators exist in all EA variants and AOS aims at the concurrent and adaptive use of several operators.

Adaptive AOS mechanisms mostly model the problem as a multi-armed bandit (MAB), a simplified version of the reinforcement learning problem [263]. Maturana et al. [198] discuss a comprehensive view of AOS, identifying the components of an adaptive operator selection framework. A credit assignment component uses feedback provided by the EA to calculate scores of operators. These scores are maintained in a credit registry and are used by the selection component in order to select the next operator to be used by the EA. In a more generic view, the AOS mechanism can also be complemented by a component that creates and adds new available operators to the AOS or removes existing ones.

The simplest operator selection method is probability matching, i.e. *softmax* selection of operators based on assigned scores. An operator's probability to be selected is calculated as the proportion of the operator's current score to the total sum of scores of all operators. This method was applied by Thierens [267] to GAs, by Gong et al. [112] to DE (the PM-AdapSS-DE algorithm) and by Qin and Suganthan [225], Mallipeddi et al. [192] and Xie et al. [290] as part of the SaDE, EPSDE and ILSDEMO ensembles described in Section 3.3.

A disadvantage of the probability matching approach stems from the allocation of probabilities directly proportionally to the rewards; it results in the best known operator not being exploited maximally because sub-optimal operators are also applied. The smaller the difference between scores the more often sub-optimal operators will be applied instead of the best known. Adaptive pursuit was used by Thierens [266, 267] to solve this problem. It increases the selection probability of the best known operator while decreasing all other probabilities.

A third selection method is the Dynamic Multi-Armed Bandits (D-MAB) suggested by DaCosta et al. [58]. The Upper Confidence Bound (UCB) algorithm is used, which will usually select the option that has the highest expected reward while still maintaining a small probability of selecting worse options for exploration purposes. However, UCB is not appropriate for a dynamic environment: if an option becomes less efficient than another it will take a lot of time for the corresponding probabilities to adjust accordingly. For this reason, a Page-Hinkley test is used to detect changes in the EA and restart the UCB with all operators being equal.

Credit assignment is usually performed by counting the number of successful applications of an operator, where success is defined as creating offspring that are fitter than the parent(s). This method is followed by most AOS mechanisms in literature.

The EXtreme value-based Adaptive Operator Selection (ExAOS) by Fialho et al. [94] uses a credit assignment method based on a sliding window. Whenever an operator is applied, the fitness improvement is calculated and added to the window. The credit assigned to the operator is the maximum value found in the window. This approach aims at rewarding operators that contribute with large improvements, even if they only do so once. It is interesting to note that this contradicts Gong et al. [112] who concluded that averaging past rewards is preferable. The ExAOS credit assignment method is paired with D-MAB selection [58] described above. A sliding window is also used by Fialho et al. [95]. They suggest increasing the reward with the time elapsed since the last application of this operator and decreasing the reward with the number of times the operator has been applied within the window. The aim of this method is to adapt quickly to (even subtle) changes of the dynamic environment. Li et al. [179] suggested a sliding window that stores the rate of improvement in the fitness of the offspring as compared to the parent. The sum of all these rewards in the window is used by a ranking mechanism to assign credit to the operators.

A different credit assignment mechanism is Compass suggested by Maturana and Saubion [200]. Based on the concepts found in [201] (see Section 3.4), an operator's impact is evaluated using measures of both fitness and diversity in order to calculate the exploration-exploitation balance achieved by the operator. The assigned credit reflects how closely the achieved balance is to an exploration-exploitation balance that is required by a user-defined schedule. Other tested credit assignment methods are based on domination between operators and Pareto fronts. Compass was paired with probability matching selection but was also combined with D-MAB in [197].

Except for the widely used multi-armed bandit approach discussed so far, AOS has also

been treated as a full reinforcement learning problem by Sakurai et al. [237], Chen et al. [50] and Pettinger and Everson [223]. Unlike the previous approaches, these methods include the notion of state that is defined using feedback from the EA. For each distinct state, separate preferences are learned for each operator and selection of the operator to apply is based on the current state of the search.

An important issue with operator selection (as with all adaptive parameter control) is the feedback used for control. Veerapen et al. [278] presented and compared various utility measures for variation operators. These measures are based on a combination of exploration and exploitation measures and use Pareto-dominance to evaluate operator utility. In another study, Whitacre et al. [285] make a distinction between the source of feedback and any further statistical treatment (a notion further elaborated in [158]). Several feedback sources are suggested (including whether the offspring survives or the number of generations it remains in the population). The data received by these sources can be treated by averaging them or by detecting the outliers in the sample. The latter method intends to award operators that produce "extraordinary" individuals. Experiments show that the choice of feedback source has a significant impact on performance while the outlier detection method is shown to be very promising.

Operator selection has also been approached in a self-adaptive manner. Spears [258] suggested a simple self-adaptive operator selection mechanism for GAs where chromosomes are extended with one bit that selects between two operators. A local approach is followed, i.e. every time a new individual is created the operator bits of the parents determine which crossover is applied. Experiments showed that the performance of the self-adaptive GA is, in most cases, similar to the best performing static GA (using one of the two operators). Control experiments, however, revealed a very interesting fact: the mere availability of multiple operators was adequate while actually self-adapting the choice of operator did not have any additional benefit. In similar work, Riff and Bonnaire [229] extend each individual with a gene denoting the operator used to create it. When a new offspring is produced, the operator of the fittest parent is employed.

Instead of including the operator identifier in the genome, an alternative is to include probabilities for all available operators. Gomez [111] implemented self-adaptive operator selection in this manner with the Hybrid Adaptive Evolutionary Algorithm (HaEa). He argued that centralized adaptive operator selection has the disadvantage of the complexity related to calculating the operators' global value while self-adaptive operator selection suffers from the need to define meta-operators for evolving genes that encode operator rates. The

latter problem of self-adaptation is tackled here by avoiding meta-operators for the genes that encode operator rates. Instead, random rewards (or penalties) are added to operator rates when the offspring is better (or worse) than the parent (in case the offspring is worse, the parent genome is kept but with the modified operator rates). Operator rates are used by a roulette selection scheme to decide which operator is used every time a child is produced.

Along the same lines, Montero and Riff [209, 210] propose operator selection self-adaption where the genome is extended to include the probabilities of the operators. The probability of the applied operator is given a reward/penalty which is either random (the "light-weight" version) or depending on the ratio of the difference between child and parent's fitness to the best/worse operator result in the last generations. An adaptive version was also suggested that maintains one global probability per operator and, at each generation, one single operator is picked to produce all offspring. Success is measured as the average success over all offspring produced and the rewards/penalties are calculated as with the self-adaptive (though here there is no "light-weight" version).

3.2.2.5 Offspring sampling distribution for real encodings

One of the best known control strategies is 'Rechenberg's 1/5 rule' [226] for controlling the σ parameter, the variation of the Gaussian distribution used for mutation. Based on theoretical results with the sphere and corridor landscapes, Rechenberg concluded that the optimal success ratio should be 1 out of 5. If greater, σ should be increased and, if less, σ should be decreased. Rudolph [230] presented an analysis of this adaptation method suggesting that when used by an elitist algorithm it will lead to premature convergence.

An extension of the 1/5 rule with reinforcement learning was suggested by Muller et al. [212]. The temporal difference learning SARSA algorithm is employed to learn the appropriate action (increase, decrease or maintain the mutation step size) given the success ratio. Two possible methods of reward calculation are tested (based on the change of the success rate or the fitness). The controller showed poor results in the evaluation of this paper.

Though the 1/5th rule was important as it immediately introduced the notion of parameter control to the very first $(1+1)$-ES, the most significant innovation of Evolution Strategies was the concept of self-adaptation: encoding a parameter of the algorithm in the genome and allowing it to undergo evolution [244]. Though self-adaptation has been applied to several other parameters since (see Section 3.4), ES self-adaptation was initially used to control the

variance σ of the Gaussian distribution used to perform mutation.

Ostermeier et al. [217] argued that the original self-adaptation of mutation step sizes does not work well for small populations because of the potentially extreme values of the random mutation and proposed a "derandomized" approach. The concept of an efficient evolution path motivated the use of correlation between individual mutation step sizes of the dimensions of the problem and the adaptation of a covariance matrix [218, 124]. Based on this, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and its variants [125, 123, 122, 119, 126, 31, 16, 146] have become very successful numeric optimizers.

A deterministic control of the mutation step size is described by Krink et al. [171]. The parameter takes values from a sequence acquired by a nature-inspired method (the sandpile model). This is part of a combined control method and is presented in detail in Section 3.3.

While discussing control of the mutation step sizes for ES, we mention the works by Rudolph [231] and by Arnold and MacLeod [14]; both contain comprehensive reviews of such methods along with their mathematical bases.

Though the parameter addressed in this subsection concerns mostly Evolution Strategies, some examples are found in other EA variants as well. Real-coded Evolutionary Programming involves an identical parameter that regulates the variance of the normal distribution used for mutation. Self-adaptation of this strategy parameter was introduced to Evolutionary Programming by Fogel et al. [98]. An application of Gaussian noise to a binary encoded GA with self-adaptation of the mutation step size was suggested by Hinterding [133]. Gene values are first decoded into numbers, noise is added and then encoded back into bit strings. Individuals are extended with one gene to encode the mutation step size and the sequence of mutations (first the step size gene is mutated with a fixed deviation then the new step size is used to mutate the other genes) is similar to Evolution Strategies.

3.2.3 Selection

A number of control mechanisms have been proposed for dynamic selection in an EA.

Several deterministic methods draw inspiration from the Boltzmann distribution which also underlies Simulated Annealing as introduced by Kirkpatrick, Gelatt, and Vecchi [164] and originates from condensed matter physics. In the case of combinatorial optimization this distribution in combination with a thermal equilibrium guarantees to have asymptotic convergence to global optima. The idea is to control selection in a deterministic way using a function that increases the selectivity over time. Goldberg [107] was the first to create a tour-

nament selection in genetic algorithms that results in the Boltzmann distribution, and as a consequence he was able to prove convergence to global optima. De la Maza and Tidor [65] focused on faster convergence and introduced a scheme which uses a Boltzmann weighting strategy for selection. They utilize a deterministic scheme which increases selectivity in a linear fashion. Dukkipati et al. [68] defined specific Cauchy criteria for a scheme to control the Boltzmann selection. Their main criterion is that the differences between successive selection pressures should decrease as the evolutionary process proceeds.

Next to approaches that focus on obtaining a Boltzmann distribution to guarantee a good outcome, several strategies have been introduced that use a similar scheme to deterministically adjust the selectivity as the evolutionary process progresses.

Marin and Sole [194] used a similar temperature-based schedule to control selection in combination with extinction dynamics. They create a graph-based space for the population in which individuals are placed and connected via weights representing differences between fitness values. Selection is based on the weights and replacement is done either at random or by selecting the best. The choice is made probabilistically with the probability depending on the temperature of the process. The result is a more likely insertion of the best individual in the end of a run.

Next to deterministic control, adaptive mechanisms have also been suggested that monitor the state of the process and adapt the selection accordingly. Affenzeller [3] introduced The Segregative Genetic Algorithms (SEGA) that uses a fixed population size and the amount of offspring generated depends on the population diversity. It was extended by the same author to SESEGASA [4] adding a method to detect premature convergence and a more advanced selective pressure mechanism. McGinley et al. [203] used the diversity of the population to adjust the tournament size, resulting in a bigger tournament size in case of a high diversity and smaller in case of a low diversity.

Another category of adaptive methods utilize some kind of spatial structure to enable the adaptation of the selection mechanism. Mass extinction is applied to obtain adaptive selection using a spatial structure based on the concept of self-organized criticality and the sandpile model in the work by Krink and Thomsen [172]. They organize individuals in a grid corresponding to the lattice of the sandpile model (for more information see [25, 24]). As grains are dropped on the grid, avalanches determine which zones in the grid become extinct. In these zones new individuals are placed which are mutations of the currently best individual. Rudolph and Sprave [232] present an approach consisting of a genetic algorithm which is placed in a particular cellular space. Each individual has certain neighbors assigned

and is only allowed to mate with those neighbors. In order for new individuals to be accepted into the population, they need to be better than a certain threshold which is dynamically controlled.

An adaptive approach inspired by ants was presented by Kaveh and Shahrouzi [160]. Their idea is to maintain a separate population (called the colony) of individuals that were shown to be promising in order to maintain population diversity. Individuals are selected and removed from the colony using a pheromone-based scheme and parents for the next generation are selected using a combination of the colony and the regular population.

Self-adaption of selection has been proposed as well. Eiben et al. [79] self-adaptated the selective pressure by encoding the tournament size in the genome. Because it is a global parameter, the value is determined by summing up all the individuals' values. In similar work, Maruo et al. [195] suggested using a majority vote instead of a summation.

On the other hand, Smorodkina and Tauritz [255] proposed self-adaptive mate selection. Each individual encodes its own preferences as a selection mechanism in the form of a tree (similar to GP). The first parent is selected via a regular selection method; it then uses its own strategy to select its mate.

3.2.4 Fitness function

Although not common in all application domains dynamic fitness functions have been successfully used for various problems. Here, we present methods for controlling aspects of the fitness function in two categories:

i. Methods that adapt the weights of penalties in constrained optimization.

ii. Other strategies applicable to unconstrained problems.

3.2.4.1 Constrained problem approaches

A straightforward strategy is to gradually increase the weights of unsatisfied contraints with time. Joines and Houck [150] suggested a mechanism that increases penalty weights as generations elapse. The Genecop II by Michaelewicz and Attia [206] makes such increases only with every restart of the search. Similar schemes were presented by Kazarlis and Petridis [161] and Ben Hamida and Schoenauer [118].

Alternating increase and decrease of penalty weights was suggested by Bean and Hadj-Alouane [29]. A weight vector determines the penalty of certain candidate solutions. The

values of this vector start very small, increase at first, and then decrease and increase in an alternating scheme (where the rate of increase is higher than the rate of decrease).

The Stepwise Adaptation of Weights (SAW) mechanism by Eiben and van Hemert [83, 82] adjusts the weights of the penalties in periods of a certain number of fitness evaluations. The constraints that are violated by the best individual in the population are identified and the corresponding weights in the fitness function are increased. Lemonge et al. [178] proposed a similar scheme but instead of considering only the best individual they take into account the entire population.

Smith and Coit [250] presented an approach which adjusts the penalty factors for constraints based on feedback regarding the severity of the constraint and the overall success of finding feasible solutions. Wang et al. [283] suggested a fitness measure which is highly dependent on the composition of the population. If the population consists only of individuals that are infeasible (i.e. that do not satisfy all the constraints), the fitness function only considers the level of satisfaction of the constraints. If all individuals fulfill the constraints, only the objective function is used as an evaluation measure. Finally, in case both feasible and infeasible candidates are present, a combination is taken. Similar methods were proposed by Tessema and Yen [265], Montemurro, Vicenti and Vannucci [208] and Farmani and Wright [85].

Co-evolving of the weights along with the solutions was suggested by Coello Coello [54]. Two separate populations are used to represent the set of candidate solutions and the weights that are used by the objective fitness function. Individuals that represent weights are applied for several generations and are evaluated by the number of feasible solutions present in the candidate solution population. Similar co-evolution but with a predator-pray flavor was used by Paredis [219].

3.2.4.2 Non-constrained problem approaches

Besides adapting weights of penalties for constrained problems, fitness functions have also been controlled to enhance performance for general unconstrained problems. Eggermont and Hemert [70] apply the SAW-ing adaptive fitness function (with some extensions) to a Genetic Programming application tasked to solve simple regression problems. The fitness measure is a weighted sum of the prediction errors for all points. The weight of each sample point is updated based on the difficulty of finding the correct value for that specific sample. The SAW-ing strategy was also applied to the field of data mining by Eggemont et al. [69].

Sakanashi et al. [236] proposed an adaptive fitness function that tries to improve the performance of GAs for the so-called GA hard problems. The fitness function adapts to make sure that the algorithm explores the search space sufficiently. This adaptation is based on the average and best performances compared with a number of generations in the past.

Efficient search is also the purpose of using multiple auxiliary fitness functions. Approaches that focus on choosing the best function using reinforcement learning have been proposed by Afanasyeva and Buzdalov [2] and Buzdalova and Buzdalov [42].

3.2.5 Parallel EAs

Parallel EAs can either be implemented as several distinct subpopulations running in parallel with occasional migrations (island model) or as several distributed evolutionary operations acting upon a common population asynchronously. In this section we present work related to:

- controlling the parameters that are particular to distributed EAs and
- control methods or ensembles for standard parameters that are designed especially for distributed EAs.

3.2.5.1 Parameters of Parallel EAs

The parameters that are specific to distributed EAs (the island model in particular) concern migration (rate, number of immigrants, policies for selection and replacement) and the topology of the islands network. Cantu-Paz [43] showed that selection and replacement policies for migration have significant effects in the convergence and takeover times of parallel GAs. The same author [44] presented a theoretical analysis of the island size, the connectivity degree between islands, the migration rate, the network topology and their interrelations. Arnaldo et al. [13] also examined the influence of the topology of distributed EAs. Using $\beta-$graphs and NK landscapes, they concluded that certain topologies (in terms of characteristic path length and clustering coefficient) are more appropriate depending on the complexity of the problem.

The simplest approach to varying the migration was suggested by Hiroyasu et al. [135]. Migration occurs in fixed time intervals but the number of individuals migrating from each island is random. The number of immigrants was also examined by Maeda et al. [191] with an adaptive approach. Again migration occurs in fixed time intervals but the number

of immigrants each island sends is adapted separately using fitness based feedback as input to a fuzzy controller. The membership functions, parameter levels and inference rules are predetermined and fixed. Instead of fixing the migration period, Lardeux and Goëffon [174] adapted the probability of occurrence of migration events. They modeled a distributed EA as a fully connected directional graph, with each edge labelled with the probability of migration occurring in the direction of that edge. When an immigrant makes an improvement in the destination island then the probability of the edge it followed is increased otherwise it is decreased.

In addition to the amount or rate of immigrants, Zhan and Zhang [299] also adapted the paths (source and destination islands), thus in effect controlling the topology of the island network. Their method sorts islands according to their mean fitness and all islands send one immigrant to every island that is higher in the ranking.

3.2.5.2 Methods for Parallel EAs

For the island model, the existence of several subpopulations running in parallel offers two important advantages: (a) there are several parameter configurations available at the same time and (b) several parameter configurations can be evaluated concurrently. The former is not directly related to control but Gong and Fukunaga [113] used it by simply setting the parameters of each subpopulation to different (even random) values. The rationale is that, at each moment during the search process, there will be at least one parameter configuration that will be somewhat favorable for further advance. Along with the effects of migration of individuals, this approach may allow for more efficient search than having only one parameter setting. The ability to evaluate several parameter vectors concurrently was used by Lis and Lis [183] to control numeric parameters of subpopulations of a parallel GA. A master is responsible for distributing populations and parameter vectors to the processors. Each parameter has a number of allowed value "levels" and at each point time only three such levels exist in any parameter vector of all islands. A run is divided in epochs of equal length (number of evaluations) for each island; at the end of each epoch, the master receives the best individuals from each subpopulation and calculates the average best fitness achieved by each level of each parameter. Then every parameter is updated by shifting values towards the best (or keeping them as are if the best was the middle one).

Control of parameters for a distributed EA with asynchronous operations acting on a common population was explored by Budin at al. [41].

A simple mechanism for controlling the crossover and mutation probabilities for a parallel GA using fitness based feedback was presented by Wang at al. [282], however, this approach is not specific to the features of a distributed EA.

3.3 Control ensembles

In this section we present work on evolutionary algorithms with combined (heterogeneous) control mechanisms put together as an ensemble that controls several parameters concurrently. These ensembles generally fall in the main categories of combining variation and population control and combining variation and selection control in order to balance exploration and exploitation. Furthermore, extensive work has been carried out for the control of the numeric variation parameters combined with operator selection for Differential Evolution (DE).

Hinterding et al. [134] combined self-adaptation of mutation with adaptive control of the population for a GA. Mutation applies Gaussian noise by first decoding the genes into numerical values and then encoding the new values back to bit strings. A gene encoding the mutation step size is added to the chromosome and mutated as in Evolution Strategies. The size of the population is controlled adaptively by maintaining three subpopulations of different sizes and dividing the run into epochs. At the end of each epoch populations' sizes are adjusted according to a set of simple rules that move sizes towards the size that performed the best during the last epoch or expand the range if the subpopulations had equal performance.

Bäck et al. [22] also combined self-adaption of crossover and mutation probabilities and dynamic adjustment of the population size. Self-adaptation is typically achieved by encoding of the operator probability values in the genome. Individual mutation rates are first mutated and the new values are used to mutate the rest of the genome. Individual crossover rates represent the probability of that individual mating. These are evaluated by random tests separately and if a selected parent is not willing to mate it creates one offspring by mutation. The population size adaptation is removed by assigning lifetimes to individuals at creation. These lifetimes are calculated based on the new individual's fitness and the population's best and average fitness. This ensemble was applied to co-operative co-evolution by Iorio and Li [147] with the difference that the crossover rates of potential parents are averaged and a common decision for mating is made instead of treating them separately.

Simple deterministic control of mutation and selection was presented by Krink et al. [171]. Both controls are based on a predetermined sequence of numbers acquired based on concepts of self-organized criticality [24]. Mutation control is performed simply by using this sequence to set the variance of Gaussian mutation. The control for selection is based on extinctions, removing a percentage of the population at each generation. These percentages follow the same sequence of numbers described earlier. Individuals to be removed are chosen randomly while the remaining individuals seed the next population. It is argued that tournament and roulette selection, near the end of the run, do not allow new (recombined and/or mutated) individuals to enter the population, while extinctions, which have strong evidence in biological evolution, do allow for this introduction of novelty.

Herrera and Lozano [131] used fuzzy logic controllers to adapt the choice of the crossover operator and the selective pressure. The aim to to maintain a good balance between exploration and exploitation; for that reason the EA employs two crossover operators, one with exploitative properties and one with explorative, with the frequency of application defined by a real-valued parameter. Selection is preformed with linear ranking and selective pressure can be controlled by a real valued parameter as well. These two parameters are controlled using two fuzzy logic controllers. The inputs are diversity-based observables (genotypic and phenotypic diversity) while no fitness-based feedback is used. The output of each controller is a delta factor to increase/decrease the corresponding parameter value.

ACROMUSE by McGinley et al. [202] is a complete ensemble with adaptive crossover, mutation and selection. Its purpose is to maintain a population that is both diverse and fit. It uses specifically designed crossover and selection operators and divides the population into exploration and exploitation sections based on diversity measures. An offspring can be created in exploration or exploitation mode according to a probability (this probability is dynamically adapted). Parent selection uses a tournament with a dynamically adapted size. The variation and selection controls are combined so that the exploring (and probably less-fit) individuals created under exploration mode with aggressive mutation are given fair chance to survive and reproduce.

In the area of DE, the SaDe algorithm by Qin and Suganthan [225] combines adaptive control of the amplification factor and crossover rate with adaptive selection between two operators. While F values are sampled uniformly from a predefined range, the CR values are sampled from a normal distribution and assigned to individual indexes to be used for a number of generations. During that period, values that create good offspring are recorded and are then used to calculate the new mean for the normal distribution used in the following

epoch. Operators are assigned credit according to the number of surviving offspring they create and are selected for each new offspring with a softmax process. SaDE was extended with multiple operators in [143] and was applied to multi-objective optimization in [141] and with separate objective-wise parameters in [142]. A study by Zielinski et al. [304] suggested that the operator selection component of SaDE is crucial to the algorithm's success and can also be beneficial when applied to two other control methods for DE.

A very similar control ensemble is used by the ILSDEMO algorithm by Xie et al. [290]. A slightly varied approach was taken by Mallipeddi et al. [192] for the EPSDE. Individuals are assigned random variation strategies and parameters in the beginning. When an offspring survives, the associated parameters are retained and added to a pool. Otherwise, new parameters are drawn from the pool or are assigned randomly. Finally, an ensemble for DE was created by Li et al. [180] by combining JADE [301] (see Section 3.2.2.3) and PM-AdapSS-DE [112] (see section 3.2.2.4).

Finally, an application specific ensemble is the Dynamic Forecasting Genetic Program (DyFor GP) model designed by Wagner et al. [281] to cope with forecasting in dynamic environments. It controls the size of the training set used and removes the population size parameter (rather replaces it with a new one). A sliding window is used for training with the historical data. The size of the sliding window is controlled on-line using two values at each step. The best model produced with each window size is evaluated using a number of future data points. The size with the best accuracy is kept while the other is moved symmetrically around the first. The population size parameter is removed using Natural Non-static Population Control (NNPC) [279], i.e. imposing a maximum limit for the sum of the nodes of all solutions in the population. This allows complex solutions of good quality to grow while keeping resource consumption within limits.

3.4 Parameter independent methods

In this section we describe generic control methods that were not designed for a specific parameter and can be applied to any (numeric) EA parameter.

A simple generic adaptive method can use a small set of values, adjust them based on simple rules and use feedback only to calculate rewards for each value (and not to make informed decisions taking into account the current situation). Wong et al. [289] proposed a probabilistic rule-driven adaptive model that maintains three possible values for each pa-

rameter. An EA run is divided into pairs of periods. During the first period values are chosen randomly (and their effect in terms of fitness improvement is recorded). In the second period values are chosen with probabilities proportional to the scores they achieved in the first period. In the end of the second period the three values are updated so as to move towards the best performing one or expand the range if none achieved performance above a threshold.

Improving the above idea, Aleti and Moser [9] suggested predicting which value would be best next based on time series forecasting, instead of using the last known winner. Continuous parameters are discretized giving a finite amount of possible values. A run is divided to iterations, at the end of each iteration the success ratio of every parameter value is calculated and added to the history list of that parameter value. Instead of simply translating these success ratios to selection probabilities for the next iteration, a line is fit to the recent history of each parameter to predict what its success ratio will be in the next iteration, thus making the selection probabilities more relevant. Credit and selection probabilities are maintained and updated for all values, making this strategy a multi-armed bandit (MAB) approach. This method was extended by adapting the discretization ranges on-the-fly: after each iteration, the best performing value (interval) is split into two while the worst performing one is merged with its worst neighbor [10].

Reinforcement learning is another problem independent approach that can be utilized here [263]. The idea is to use feedback from the EA that describes the state of the search and implement actions as changes to parameter values. Eiben et al. [73] used temporal difference learning to map states of the EA (described by fitness based metrics) to actions, i.e. parameter values. The controller algorithm uses a table that maps pairs of states and actions to estimated rewards. Learning this table is done using a combination of the Q-learning and the SARSA algorithms. When doing exploration, random (and potentially harmful) actions are chosen. For this reason, areas explored are chosen to be close to the known optimal actions. When doing exploitation, the best action for the current state is found with a separate underlying genetic algorithm that optimizes the expected reward.

A special case of generic adaptive controllers are designed to be problem specific and need to be tuned to the problem at hand to learn a mapping from feedback to parameter values (they can operate only within the top left box in Figure 2.2). Such an approach was suggested by Kee et al. [162] where the mapping is learned during a training phase while the exact mechanism implementing this mapping is a module/design choice. Two alternative methods (table-based and rule-based) are used to map a state vector composed of three metrics (fitness change and variance and population variance) to parameter val-

ues. Karafotias et al. [158] employed neural networks to map diversity and fitness based feedback to parameter values. The weights of the network are calibrated off-line using a parameter tuner. Aine et al. [5] suggested "profiling" algorithms for specific problem types by an off-line training phase. This training results in tables that map the state of the EA (described by fitness and diversity measures) to not only parameter vectors but also the time until the next update of parameter values. Lee and Takagi [177] used fuzzy logic controllers instantiated by a meta-GA. Rules are represented by encoding the centers of the membership functions (which are fully overlapping) and the ID of a rule (i.e. the combination of inputs-outputs). Fuzzy logic was also used by Maturana and Saubion [201, 199] to achieve a balance between exploration and exploitation. The correlation between parameter values and the resulting diversity/fitness is learned during an initial learning phase. After that the parameters are controlled so as to maintain a certain exploration-exploitation balance which is dictated by a user-defined schedule (thus this method, besides the automatic calibration, also requires a hand-tuning process).

Instead of designing a concrete control strategy or a method for calibrating one, Liu et al. [185] presented a platform for expressing ad-hoc control strategies through a scripting-like language (PPCea). The user is provided with several measures of exploitation and exploration derived from ancestry trees that record the whole evolutionary process as parents - children branches.

Finally, we consider self-adaptation as a generic parameter control method. It entails incorporating parameter values in the genome and allowing them to undergo evolution, relying on selection to promote good values that are piggybacked on the solutions they helped create [76]. Though self-adaptation was initially coined for controlling the mutation parameter of Evolution Strategies, it could be applied to any parameter (i.e. it is at least feasible to implement such a control). As an example, a fully self-adaptive evolutionary algorithm was presented by Maruo et al. [195]. Probabilities of mutation and crossover, the mutation step size (for real coded representations), the crossover operator and the size of the selection tournament are self-adapted. Values of global parameters (crossover type and tournament size) are decided by the majority. Self-adaptation of global parameters was also investigated by Eiben et al. [79], though here, instead of majority voting, global parameter values are calculated as the sums of all votes. To keep the votes within defined bounds, self-adaptive mutation is not used (instead the static scheme from [23] is employed). Literature reviews on self-adaptation can be found in the works by Kramer [169] and Meyer-Nieberg and Beyer [205]. These reviews convincingly demonstrate the power of self-adaptation (mainly muta-

tion parameters), especially in real-coded search spaces and in environments with uncertain or noisy fitness information. Meanwhile, they also point out that self-adaptation techniques may suffer from premature convergence and a (near-)optimal mutation strength is not always realized. This latter effect has also been observed in Artificial Life settings for digital organisms with discrete genotype spaces [52].

3.5 Trends and Challenges

Following the review of the related work, we now have covered the material to base a more informed high-level discussion of the field. In specific, we can derive some key points from the preceding survey: there are some trends evident in the field (regarding focus, direction and impact) and, more important, certain challenges that impede the research and practice of parameter control in terms of methodology, applicability and effectivity. Both are discussed in this section.

3.5.1 Trends

Figure 3.1 presents the number of publications in a histogram form in bins of 4 year periods split by the categories parameter specific (that is further split by parameter), ensembles, and generic. A plot of the total amount of publications on a cumulative scale is shown in Figure 3.2. It seems that the interest in parameter control modestly increases, while the drop at the end can be attributed to the dissemination delay of new publications.

As we can see parameters belonging to variation operators have received by far the most attention. Population related parameters are much less popular and parameters related to selection have received even less attention. Furthermore, ensembles for combined control of multiple parameters and generic control methods that can be applied to any parameter have been explored to very little extent (but they seem to gain traction lately).

The modest number of generic control methods implies what we call ‘the patchwork problem’: if one is to control more (all) parameters of an EA, then for each parameter she has to choose from a parameter specific set of existing methods and mix them into one system. Unfortunately, little is known about the joint effects of control mechanisms, thus there are no good guidelines about how to create good combinations. Therefore, the resulting mix is necessarily ad hoc and likely suboptimal. The work of Bäck et al. [22] is a good illustration for this as it mixes 3 different mechanisms to control 3 different parameters

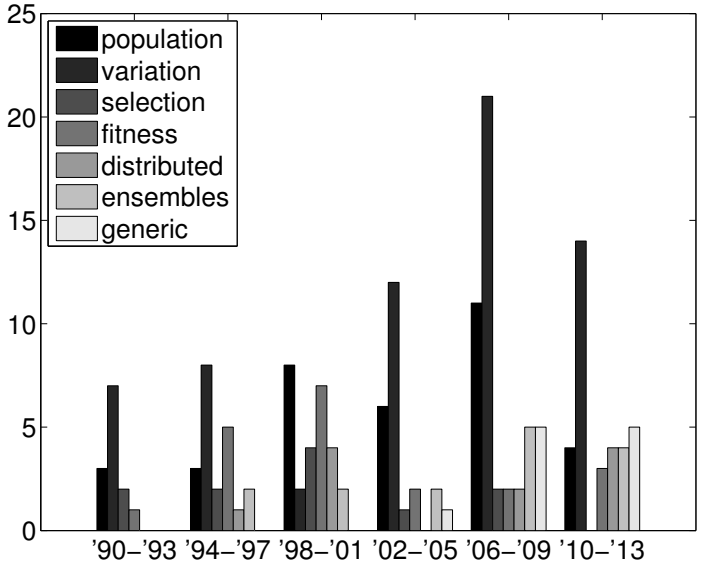


Figure 3.1: *The number of publications in 4 year periods for each parameter category described in this chapter. For each period, bars denote (from left to right) population, variation, fitness, selection, distributed EAs, ensembles, generic. NB. Data for 2013 concerns the January-May period only.*

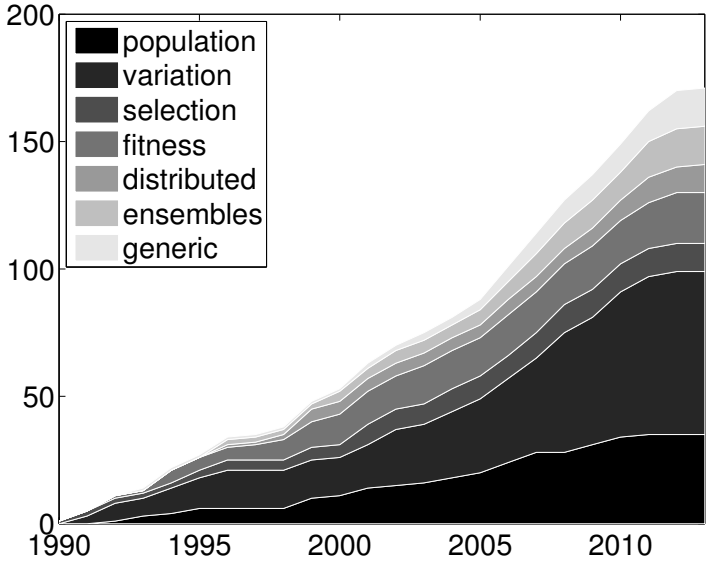


Figure 3.2: *The total number of publications on parameter control on a cumulative scale after 1990. Greyscale levels indicate the parameter addressed in the papers.*

in an eclectic system without much justification. An encouraging exception is the ensemble by McGinley et al. [202] which is well orchestrated with controllers designed to work in cooperation.

Considering the methods and mechanisms used for adaptive parameter control, we can identify several categories that are relatively frequent in the literature:

- *Formulas* that calculate a parameter value using certain feedback from the EA. Such formulas may be based on theoretical results (e.g. [254], [107]) or the intuition of the designer (e.g. [260]).

- *Rules* triggered by certain events/thresholds concerning feedback from the search. They specify responding actions to these events, usually according to the designer's intuition (e.g. [75]).

- *Fuzzy Logic controllers* whose input is feedback from the search. Their output may be used to trigger hand-coded rules (in which case there is an overlap with the above category, e.g. [184], [191]) or they may directly output parameter values (e.g. [131]).

- *Value Sets* that are adapted, i.e. multiple (two or three) values are tested and the best one is used in the next phase while the others are adjusted accordingly. The testing phase is then repeated again (e.g. [183] [289] [281]).

- *Multi-Armed Bandit* strategies that treat each value as a separate arm and adapt their selection probabilities by learning expected rewards. This approach is very popular with Adaptive Operator Selection (e.g. [267], [58]).

- The full *Reinforcement Learning* approach that uses feedback from the search as descriptors to define states and maps actions to parameter values (e.g. [73], [212], [2]).

Regarding the field of parameter control in EAs as a whole, we can note a disappointing lack of impact. To be specific, many of the related papers report promising experimental results, however, no parameter control method or strategy has been widely adopted by the community or has became part of the common practice toolkit of evolutionary computing – a problem that was also noted by De Jong [63]. Perhaps this is caused by the lack of convincing evidence of the added value of control techniques. Putting it differently, despite the existence of important pieces of related work, it seems that most of the papers published make but a limited contribution to the field. All too often, such a paper (including ours!) has a limited scope and focuses on a simplistic control of a single specific parameter based on the author's intuition of how the parameter should vary. Typically, the paper does not analyse and explain the resulting behaviour of the parameter and the EA, nor does it properly evaluate the added value of the control strategy based on good benchmarks.

3.5.2 Challenges

The first, and perhaps most relevant, challenge is identifying the niche for parameter control in general, and that of specific parameter control mechanisms in particular. In other words, we should try to understand for which problems and EA performance requirements is a given method advantageous. A related question we frequently receive is: "Is parameter control better than parameter tuning?", or "When should I do parameter tuning and when should I do parameter control?". For a solid answer based on quantitative evidence there is a lack of experimental data (with very few exceptions e.g. [101]) and –even more importantly– a lack of a decent research methodology.[5] A possibly useful angle to this end is to consider the application scenario as an important dimension. Using the categories of Eiben and Smith in [81], Chapter 13, it could be argued that for repetitive problems parameter tuning is worth the extra effort, whereas parameter control is the logical choice for one-off problems, applications with dynamic fitness functions, and on-line adaptation, e.g. robotics [67], [117]. Nevertheless, extra research and discussions are needed to back up solid answers to this and similar questions.

A relevant issue is that of *meta-parameters*, i.e. the parameters of the controllers themselves, on which the whole concept of parameter control relies. There is a widespread opinion among parameter control researchers that even though a parameter control mechanism introduces new parameters of its own, the extended system (EA + parameter controller) is less sensitive to these parameters than the original EA to its own parameters - otherwise the parameter controller would be pointless. The skeptical observation, however, is that there is little real evidence to back up this belief, with the σ's and τ's in evolution strategies being the only well-known exception. A natural way to address this issue is to study parameters of parameter control mechanisms. For instance, using sensitivity analysis [238] can help verify or refute the aforementioned fundamental assumption. Furthermore, these studies can lead to better control mechanisms, simply because their working *does* depend on their parameters, even if a control mechanism is more robust than the bare-bone EA it regulates.

Regarding the research and development of better control methods, we identify two important challenges. The first is directly derived from one of the trends and obvious: working on reducing the patchwork problem mentioned above. This can be addressed by carrying out investigations about the combination of different control mechanisms and trying to under-

[5] The weakness of experimental methodology is in fact a problem for the whole EC field, as noted by Eiben and Jelasity [74].

stand their joined effects. Additionally, more research could be done on controlling parameters that have received relatively little attention in the past. Of course, developing generic control mechanisms that work on any –or at least many– parameters could be most helpful here. The second is identifying relevant EA behaviour descriptors (sometimes called observables or monitors) which can offer direct advantages for developing better parameter control mechanisms. In particular, it can help designing mechanisms for adaptive control, because these are based on using information (feedback) from the evolutionary search process. Being able to identify the most relevant information (EA behaviour descriptors) can obviously improve the decisions made about new parameter values based on that information. This research line could benefit from data mining techniques that disclose the correlations between various EA behaviour descriptors over time and the links between such descriptors and (the online dynamics of) solution quality.

In terms of better research methodology, progress could be made along several lines as well. For instance, the EC community should agree on sound measures for 'effort' in order to make fair comparisons between different techniques. This is a nontrivial problem, because the tuning effort occurs during the design stage before the real EA run, while the computational overhead caused by control mechanisms manifests itself during a run. Furthermore, while there are software and hardware independent measures for tuning effort, these are based on counting fitness evaluations, see e.g. [247]. However, the extra work that parameter control mechanisms perform may be hidden from such a counter. Using computing time for this seems an easy solution, but that may raise other issues, as discussed by Eiben and Jelasity in [74]. Another essential prerequisite for sound assessments of control techniques is a good benchmarking practice. This issue reaches further than the composition of a good test suite. The added value of a parameter control technique will prove different when comparing it to different alternatives. This is a relevant and nontrivial question, because there are several justifiable options here, including:

- Comparison with the same EA without parameter control
    - with commonly used ('default') parameter values, or
    - with optimised (tuned) parameter values.
- Comparison with the same EA with a blind control mechanism, i.e. randomly varying the parameter values, see Karafotias et al. [154].

– Comparison with the same EA using another mechanism to control the same parameters.

Which of these and other possible options is appropriate for a good assessment should be investigated and widely discussed.

Last but not least, let us mention the admittedly rather general issue of a deeper understanding of parameter control mechanisms. Though it might seem simpler at first glance, properly analysing and interpreting the behaviour and results of a parameter control mechanism can pose a considerable challenge due to the multiple levels of interaction, the stochastic nature of the process and the obscurity and lack of understanding of the underlying evolutionary algorithm itself. Very little has been in this direction so far: similarly to the main body of work within EC, most publications about a parameter control mechanism simply show *that* it works. Usually, there is no or very little discussion about *why* it works and *how* it works. In this respect, major improvements are possible through developing the know-how of monitoring the dynamics of the controlled parameter(s) and the whole EA over time. This requires the identification of relevant behaviour descriptors for EAs, such as the extent of exploration and exploitation (a concept that is itself difficult to define but very relevant [277], [78]), population diversity, solution quality, and many more. Such improvements will also contribute to the design of more powerful adaptive controllers as was explained in our second suggestion above.

Part II

Algorithms & Experimental Results

4

Generality and Applicability

So far we've explained the problem of parameter setting in evolutionary algorithms and introduced a theoretical framework that relates tuning and control, decomposes and classifies control mechanisms, and maps the parameter control problem to reinforcement learning. We have also reviewed the existing work in the field and identified the existing trends and some important challenges. We are now able to introduce, motivate and position a more concrete investigation into the subject. The second part of this thesis is devoted to presenting algorithmic designs of parameter control mechanisms, experimental results on their performance and analyses of their behaviour. Furthermore, it addresses the question of the fundamental and practical value of parameter control by experimentally evaluating two of the arguments supporting parameter control: (i) different parameter values are better at different points in an evolutionary process and (ii) parameter control is especially useful when dealing with dynamic problems (see Section 2.2).

The survey of the field of parameter control revealed two important trends (Section 3.5): (i) there is a lack of impact and adoption of parameter control in common practice and real world applications and (ii) the majority of control methods implemented and published so far

are parameter and/or EA specific. We believe that there are multiple factors that contribute to the former with the latter being one of them. The EAs used in research of parameter control are not necessarily employed in real world applications. On the contrary, it is often the case that real world problems, due to complexity or other requirements and limitations, demand a different approach with often problem-specific representations and corresponding operators. Furthermore, businesses often develop tools that use proprietary algorithms that can be very different to those employed by researchers. Therefore, we believe that parameter control needs to become detached from the EA and problem of the application, the same way that the evolutionary algorithm paradigm is independent of the problems it solves.

The algorithms that we present in the following chapters fall in the category of generic controllers: they are independent of the EA, its parameters and, of course, the problem it (the EA) is solving. They can be seen as plug-ins that can be readily employed by the designer/user of an evolutionary algorithm without the need to make any changes or adjustments to the EA. The advantages of such an approach in terms of applicability and usability are obvious. Of course, one can argue that such generic add-ons will have sub-optimal performance; given the established view of the heuristics community regarding black box and specialised optimisers and theoretical results such as the No Free Lunch theorem [288], it is a valid argument. Nevertheless, we think that a readily usable control mechanism that can still offer substantial performance improvement is more valuable, especially for real world applications where the EA designer/user does not have the time or the expertise to create a specialised controller; a generic component for parameter control fits a generic black box solver.

A second, and perhaps more important, factor that contributes to the lack of practical adoption of parameter control is that its innate value itself has not been investigated thoroughly enough. We think that more and more focused work is required to address and evaluate the specific arguments supporting parameter control (see Section 2.2); existing experimental work mostly falls in the category of a horse race between an arbitrary (often ad hoc) EA and its "adaptive" version for a test problem, making a less convincing argument in favour of parameter control in general and its potential in real world applications. In the experimental work of this thesis we make an effort to counter this problem by carrying out tests specifically designed to evaluate a specific argument motivating control and by choosing a collection of "relevant" (well-established and competitive) EAs and, when possible, problems from real world applications when assessing our control algorithms.

The experiments and algorithms discussed in the subsequent chapters address the follow-

ing subjects. First, we introduce concrete algorithmic designs for generic parameter control, evaluate the performance gain using competitive EAs and relevant problems and analyse their behaviour. Both categories of control (Figure 2.3) are examined: off-the-shelf control is treated in Chapter 6 and tuned control is investigated in Chapter 7. Second, we assess two of the supporting arguments motivating parameter control: the advantage of having different parameter values at different stages of the optimisation process is explored in Chapter 5 while the benefit of parameter control in a dynamic environment is tested in Section 6.5 for off-the-shelf generic control and is a key concept in the mixed approach examined in Chapter 8.

The research questions that we will try to answer experimentally in Part II are the following:

i. Is an off-the-shelf generic parameter controller a viable idea that can offer benefits in the performance of an EA without the need of calibration to the problem at hand?

ii. Is a tuned generic parameter controller a viable idea that can improve the performance of an EA by calibrating the control policy to suit the problem being solved?

iii. Is there an inherent advantage in having different parameter values at different times during an evolutionary process?

iv. Is parameter control advantageous when facing dynamic problems?

5

Varying Parameters and Randomness

In this chapter we investigate the view that different parameter values are better at different points of an evolutionary run. It is a fundamental argument motivating parameter control: an inherent benefit in changing parameter values provides a solid justification for control. Thus, it is a fundamental question whether such inherent benefit exists. This question may seem trivial for some parameters, e.g. the mutation step size σ of evolution strategies: it is generally accepted, and often practically demonstrated, that "σ should start with high values and converge to small values as the search converges to the optimum" [81]. Nevertheless, for other parameters such understanding does not exist and it is the standard practice to keep their values fixed. The only study we are aware of that directly addresses this question is found in [56], where deterministic schedules are used to investigate if there is an intrinsic advantage in having dynamic population sizes. However, we believe that the existence of a schedule already introduces a strategy and, thus, the effect of parameter value change alone is not actually isolated. Instead, the goal of the experiment in this chapter is to investigate whether (non-intelligent) 'variation' alone is sufficient to improve EA performance. To this end, we implement a few simple random methods to vary parameter values during the run of an EA and investigate their impact on a set of standard benchmark prob-

lems. In particular, we use a uniform distribution and a Gaussian distribution and compare the resulting EAs with an EA whose parameter values are fixed (by a powerful tuning algorithm) and with an EA whose parameters change by a sine wave based schedule (enabling increasing and decreasing the values).

Additionally, we have a second, alternative motivation that is rooted in methodology. In many of the studies introducing a parameter control strategy, performance comparisons between the EA using the control mechanism and the equivalent EA with static parameter values are presented as a proof of the controller's value. However, it is the usual case that no further investigation is carried out as to how exactly the parameters are varied and to what extent the performance gain is a result of the specific control strategy or the mere fact that parameters simply change during the run, i.e. just adding some variation in the parameter values already brings added value. The idea that simply changing the values of a parameter, regardless of how that change is done, can result in better performance has been hinted in some previous work. In [258], Spears experiments with self-adaptive operator selection for genetic algorithms. Results show that the GA with random operator selection has similar performance with the self-adaptive GA meaning that it is just the availability of multiple operators that improves performance and not self-adaptation. Randomised values are purposefully used in [113] to set the parameters of different islands for a distributed EA with the rationale that, at each moment during the search process, there will be at least one parameter configuration that will be favourable for further advance. Finally, random sampling has also been employed to control the variation parameters of Differential Evolution [301], [290], [225].

The following sections describe the experiments we carried out in order to answer the question whether a random variation of parameter values by itself (with no intelligence, purpose or strategy) can have a positive effect on the performance of an evolutionary algorithm. Though theoretical studies on optimal parameter values or control strategies do exist (see e.g. [132], [175], [148] and [186]), we believe that such a theoretical approach here would be impossible or greatly oversimplifying. For this reason we prefer an experimental approach as will be described in the following section.

5.1 Experimental Setup

The purpose of the experiments presented here is to determine if there can be an intrinsic merit in the mere variation of parameter values (with no particular method or strategy) in terms of performance gain for the evolutionary algorithm. In order to assess the effect of parameter variation isolated from the effect of an "intelligent" control method or strategy we use the most naive parameter variation approach possible, i.e. random variation of parameter values. Keeping all other factors identical, we compare the performance of an evolutionary algorithm when its parameter values are kept fixed during the whole search and when its parameter values vary according to some random distribution. To show the difference between the random variation and a non-random (but certainly not sophisticated) variation approach, an additional approach to vary the parameter values, i.e. a sine-based function, is used which facilitates sequences of increase and decrease of such values.

As an EA we use a Simple Evolution Strategy (SES) solving a set of standard continuous optimisation test functions (for detailed descriptions of the EA and the problems refer to Appendix B). The SES has a fixed crossover type (uniform) and selection type (plus selection); the other parameters are subject to tuning and control in this experiment: the population size μ, the generation gap g, the mutation step size σ, the number of crossover points N and the sizes of the parent and survivor tournaments k_p and k_s respectively (the ranges of tuning and control[1] for each parameter are shown in Table 5.1). The test functions we use are:

- f_1 Ackley
- f_2 Rastrigin
- f_3 Rosenbrock
- f_4 Schaffer
- f_5 Bohachevsky
- f_6 Griewank
- f_7 Shekel

[1]These ranges can be different than the domains of the parameters listed in Appendix B, e.g. for parameters with open infinite theoretical domains, we chose ranges of tuning/control we consider reasonable.

Several parts of the experiments in this Chapter involve a tuning process. In all these cases we have employed Bonesa [248] as a parameter tuning method. Bonesa is a model-based algorithm composed of two intertwined loops for searching and learning. It learns a model (based on nearest neighbours with relative distances, and Gaussian filtering) of the quality of parameter settings and uses this model to direct the search in the parameter space. New information acquired from the search process (results from testing new parameter settings) is used to further refine the model. Experimental results [248] have shown that Bonesa stands competitively among the state-of-the-art parameter tuners. Each parameter tuning process entails a number of runs of the EA being tuned (as part of the search process). Of course, a higher number of such runs can help improve the results of tuning. This "budget" of EA runs will be specified for each tuning process performed in the experiments here.

We use the following workflow to facilitate the desired comparison, in steps (2)-(4) experimental results are generated and comparisons are made:

(1) Tune the parameters values of the SES, resulting in a set of *basic* parameter values.

(2) Add variation to the *basic* parameter values found under (1) using a gaussian and uniform distribution with fixed *variation* values.

(3) Instead of using fixed *variation* values as in (2), try to find the best *variation* values using a parameter tuning approach given the *basic* values found under (1).

(4) Tune *all* parameter values (both *basic* and *variation* values) at the same time, also include a non-randomised parameter value generator which can express basic sequences of increasing and/or decreasing values.

Each of these steps is discussed more elaborately below.

5.1.1 Tuning the SES

As a first step, the SES is tuned for every test function separately (all six parameters are tuned concurrently with one tuning process per problem). This step results in seven parameter vectors $\vec{p_i}$, one for each problem $f_i, i = 1...7$, with good static values for each parameter. The ranges and the results of the tuning process are shown in Table 5.1. A single Bonesa tuning run was given a budget of 10000 EA runs.

Table 5.1: *Parameters and corresponding tuning ranges and tuned values for each problem.*

	Range	f_1	f_2	f_3	f_4	f_5	f_6	f_7
μ	$[1, 200]$	199	147	190	150	5	186	195
g	$[0, 15]$	0.137	4.062	0	1.826	0.001	0.327	0.114
σ	$[0, 2]$	1.053	0.008	0.09	1.736	1.991	1.808	0.037
N	$[1, 9]$	6	6	6	5	6	7	6
k_p	$[1, 200]$	21	2	32	3	4	2	121
k_s	$[1, 200]$	163	16	3	9	196	6	127

5.1.2 Experiment 1: Adding variation around tuned values

In order to determine the effect of variation we use the tuned vectors $\vec{p_i}$ as a starting point and add some random variation. At each generation, parameter values are drawn from a distribution (gaussian and uniform distributions are tested). For each parameter, a separate distribution is used and the "centers" of these distributions are set to the tuned values:

- *gaussian*: for problem i, values for parameter j are drawn from a normal distribution $N(\vec{p_i}(j), d \cdot \vec{p_i}(j))$
- *uniform*: for problem i, values for parameter j are drawn uniformly from the interval $[\vec{p_i}(j) - \frac{w}{2}, \vec{p_i}(j) + \frac{w}{2}], w = d \cdot \vec{p_i}(j)$

Several width coefficients d are tried, $d =$ 0.01, 0.02, 0.03, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5, 0.8. Separate runs are made with each parameter varied alone and all parameters varied together.[2] For every setting (i.e. combination of parameter, distribution and d), the SES is run 30 times to derive statistically reliable results.

5.1.3 Experiment 2: "Tuning" the range of the variation

The above process attempts to determine whether adding some variance around the parameter values can result in improved performance, however, only a small number of hand-picked ranges (defined by the values of d) are tested. For a more thorough and rigorous test, the rationale of experiment 1 is maintained but we use Bonesa to tune the standard deviation of the gaussian distribution. Thus now, for problem i, values for parameter j are drawn from a normal distribution $N(\vec{p_i}(j), \sigma_i^j)$ with every σ_i^j being derived through a search process by

[2] Varying all parameters together is an obvious choice, considering the interaction between parameters. We did not include settings where only subsets of specific parameters are varied as that would increase the overall size of the experiment too much.

Bonesa (one tuning process per problem was executed that concurrently tuned the deviations of all six parameters). A much longer (25000 EA tests) tuning process was used for this experiment to increase the reliability of the results. Due to time limitations, this experiment was performed for only one function, we arbitrarily chose f_1.

5.1.4 Experiment 3: "Tuning" all the settings of the variation

As a final test we make a *fair* comparison between the performance of the SES using static parameter values and its performance using varying values. Since the static values were derived through a tuning process, in order to make the comparison fair, the settings that determine the varying values must be calibrated as well. Thus, an identical tuning process (using Bonesa with the same budget of 10000 EA tests) is used. Here, except for the normal and uniform random distributions employed previously, an approach based upon a sine wave which is able to generate sequences of increasing and/or decreasing parameter values is used as well. For each variation mechanism, the following settings are tuned:

- *gaussian*: for each problem i, a tuning process calibrates for each parameter j the mean m_i^j and standard deviation σ_i^j of the normal distribution
- *uniform*: for each problem i, a tuning process calibrates for each parameter j the minimum l_i^j and the width w_i^j of the range from which values are drawn
- *sine*: for each problem i, a tuning process calibrates for each parameter j the amplitude A_i^j, frequency f_i^j, angular frequency ω_i^j and phase ϕ_i^j that define a sine wave used as a deterministic schedule

After the tuning is complete, for each problem and variation setting combination, the SES is run 30 times to derive statistically reliable results.

5.2 Results and Analysis

The results of experiment 1 are presented in Tables 5.2 and 5.3. These tables show the performance of the three SES variants that have been run in this first experiment: with tuned static parameter values, with fixed noise from a gaussian around these static values and the same for the uniform distribution. Underlined values indicate a significant improvement

Table 5.2: *Results of experiment 1 with Gaussian sampling.There is a subtable for every function; for every line it is denoted which parameter is varied. The first column of each subtable shows the performance when the parameter is kept static and the subsequent columns show the performance when the parameter is varied with the corresponding value of d. All numbers are averages over 30 runs. Emphasised values show performance that is significantly better than static (with 0.95 confidence). Lower values are better for all functions.*

f_1 (Ackley)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		3.22	2.75	3.07	2.72	2.78	2.65	2.75	2.81	2.69	2.70
g		2.72	2.81	2.73	2.78	2.75	2.77	2.71	3.20	2.80	2.73
σ		2.69	2.73	2.78	2.80	2.65	**2.50**	**2.43**	2.41	**1.23**	**1.12**
N	2.67	2.67	2.67	2.72	2.77	2.68	2.68	3.26	3.09	2.80	2.69
k_p		2.69	2.70	2.67	2.74	2.78	2.69	2.75	2.72	2.75	2.78
k_s		2.77	2.79	2.66	2.72	2.68	2.69	2.77	2.76	3.11	2.71
all		2.82	2.73	2.68	2.87	3.20	2.54	**2.42**	**1.86**	1.59	2.00

f_2 (Rastrigin)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		8.64	7.65	7.88	7.70	8.13	7.88	7.77	9.43	12.39	31.03
g		8.30	7.53	7.51	8.05	7.74	8.34	8.46	8.37	9.34	13.99
σ		7.32	7.31	7.29	7.34	7.22	6.99	7.10	7.26	6.87	7.06
N	7.32	7.32	7.32	7.32	6.83	6.33	6.71	6.39	6.59	8.76	12.37
k_p		7.32	7.32	7.32	7.32	7.31	7.25	7.13	7.66	7.89	8.25
k_s		7.32	7.84	7.51	7.21	7.66	7.53	7.66	7.63	8.05	8.90
all		8.38	7.71	7.38	6.98	**5.71**	**5.75**	6.28	7.22	15.35	40.55

f_3 (Rosenbrock)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		9.82	7.27	8.27	7.53	7.51	7.73	7.68	8.22	7.69	8.18
g		7.56	7.56	7.56	7.56	7.56	7.56	7.56	7.56	7.56	7.56
σ		7.53	7.89	7.84	7.82	8.18	7.95	7.94	7.41	7.30	7.49
N	7.56	7.56	7.56	9.99	7.73	7.67	8.04	10.09	7.96	10.53	8.99
k_p		9.78	7.93	8.30	10.25	7.93	9.94	8.10	7.49	10.39	7.79
k_s		7.56	7.56	7.56	7.70	7.98	7.96	7.92	7.61	7.54	9.52
all		8.22	10.69	8.71	8.13	7.72	8.21	7.80	7.39	7.12	7.05

f_4 (Schaffer)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		**10.78**	11.18	10.95	11.04	11.60	11.63	11.79	12.92	15.38	30.87
g		11.52	11.23	11.26	11.22	11.39	11.58	11.18	11.52	12.38	14.40
σ		11.48	11.46	11.74	11.53	11.23	11.18	11.15	**10.63**	10.67	**9.80**
N	11.51	11.51	11.51	11.51	11.70	11.70	11.92	11.74	11.73	12.10	12.26
k_p		11.51	11.51	11.51	11.51	11.68	11.38	11.87	12.03	11.43	11.48
k_s		11.51	11.47	11.68	12.11	11.69	11.69	11.48	11.73	11.85	12.06
all		11.23	11.72	11.84	11.01	11.67	11.15	11.05	**10.63**	16.32	36.76

f_5 (Bohachevsky)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		16.16	16.16	15.80	16.04	15.61	16.65	15.67	15.52	15.64	15.83
g		16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16
σ		15.91	16.00	15.79	15.15	14.78	**13.74**	**12.21**	**7.89**	**3.95**	**3.25**
N	16.16	16.16	16.03	15.56	16.51	16.20	15.82	16.19	15.28	16.64	16.01
k_p		16.16	16.16	16.33	15.24	16.05	16.50	15.10	15.72	15.02	15.57
k_s		16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.34	15.75	17.61
all		16.37	16.59	15.76	15.24	**14.85**	**12.49**	**12.44**	**7.45**	**5.33**	**7.61**

f_6 (Griewank)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		0.81	0.80	0.83	0.86	0.89	0.79	0.94	2.35	3.17	16.68
g		0.87	0.86	0.85	0.89	0.92	0.80	0.78	0.93	0.94	1.18
σ		0.86	0.86	0.84	0.79	0.77	0.85	0.83	0.86	0.77	1.17
N	0.87	0.87	0.88	0.85	0.79	0.99	0.83	0.90	0.92	0.97	0.95
k_p		0.87	0.87	0.87	0.87	0.85	0.83	0.82	0.82	0.89	0.85
k_s		0.87	0.87	0.84	0.77	0.86	0.94	0.88	1.09	0.79	0.88
all		0.88	0.83	0.80	0.82	0.81	0.92	1.01	0.86	5.28	26.38

f_7 (Shekel)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		-1.65	-1.34	-1.30	-1.64	-1.60	-1.60	-1.57	-1.69	-1.58	-1.64
g		-2.03	-2.00	-2.03	-2.03	-2.03	-2.02	-2.03	-2.02	-1.72	-2.00
σ		-1.97	-2.00	-1.97	-1.96	-1.97	-2.00	-1.97	-1.97	-2.01	-2.01
N	-1.97	-1.97	-1.97	-1.97	-2.00	-1.68	-1.42	-1.45	-1.60	-1.80	-1.67
k_p		-1.80	-1.35	-1.35	-1.50	-1.36	-1.34	-1.40	-1.65	-1.38	-2.02
k_s		-1.94	-2.03	-2.00	-2.03	-2.03	-2.00	-2.02	-2.03	-2.00	-2.03
all		-1.96	-2.08	-2.03	-1.76	-1.40	-2.05	-1.54	-1.82	-1.78	-2.01

Table 5.3: *Results of experiment 1 with uniform sampling. There is a subtable for every function; for every line it is denoted which parameter is varied. The first column of each subtable shows the performance when the parameter is kept static and the subsequent columns show the performance when the parameter is varied with the corresponding value of d. All numbers are averages over 30 runs. Emphasised values show performance that is significantly better than static (with 0.95 confidence). Lower values are better for all functions.*

f_1 (Ackley)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		2.70	2.75	2.74	2.74	2.81	2.82	2.76	2.77	3.18	2.77
g		2.67	2.78	2.64	2.77	2.76	2.71	2.78	2.77	2.68	2.74
σ		2.74	2.80	2.78	2.78	3.28	3.27	3.19	2.67	2.63	**2.39**
N	2.67	2.67	2.67	2.67	2.67	2.74	2.74	2.68	2.67	2.81	2.70
k_p		2.67	2.67	2.75	2.65	2.70	2.70	2.67	2.70	2.80	2.66
k_s		2.70	2.77	2.77	2.66	2.70	2.78	2.77	2.79	2.74	2.70
all		2.71	2.78	2.80	2.76	2.72	2.74	3.19	2.68	2.56	**2.50**

f_2 (Rastrigin)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		7.82	7.42	8.20	7.93	8.64	8.38	8.10	7.88	8.57	8.58
g		7.87	7.95	7.83	7.75	7.45	7.41	8.60	8.20	8.36	8.76
σ		7.33	7.35	7.35	7.35	7.31	7.32	7.32	7.44	7.23	7.25
N	7.32	7.32	7.32	7.32	7.32	6.18	6.18	**5.71**	**5.94**	**5.93**	6.58
k_p		7.32	7.32	7.32	7.32	7.32	7.32	7.32	7.85	7.85	7.85
k_s		7.32	7.32	7.32	7.46	7.66	7.66	7.93	7.41	7.59	7.96
all		8.09	8.44	7.80	7.97	**5.50**	**5.54**	6.40	6.34	6.42	7.21

f_3 (Rosenbrock)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		7.85	7.61	8.02	7.96	8.35	8.43	7.94	8.03	8.09	8.51
g		7.56	7.56	7.56	7.56	7.56	7.56	7.56	7.56	7.56	7.56
σ		7.74	7.49	7.16	7.53	7.71	7.60	8.10	10.09	10.06	7.34
N	7.56	7.56	7.56	7.56	7.56	7.72	7.72	7.88	8.19	7.52	8.20
k_p		7.56	7.98	7.98	8.16	8.54	9.77	7.77	9.88	8.15	8.39
k_s		7.56	7.56	7.56	7.56	7.56	7.56	8.19	8.19	7.94	7.94
all		8.09	8.48	8.16	8.11	12.08	7.83	8.31	7.95	8.18	7.88

f_4 (Schaffer)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		11.43	11.12	11.26	11.30	11.31	11.37	11.47	11.56	11.45	11.54
g		11.71	11.74	11.98	11.78	11.77	11.19	11.64	11.40	11.58	11.34
σ		11.53	11.71	11.39	11.55	11.32	11.25	11.45	11.38	11.60	10.98
N	11.51	11.51	11.51	11.51	11.51	11.59	11.59	11.59	11.93	11.81	11.48
k_p		11.51	11.51	11.51	11.51	11.51	11.51	11.92	11.92	11.57	11.57
k_s		11.51	11.51	11.51	11.51	11.68	11.23	11.44	11.02	11.55	11.64
all		11.55	11.05	11.66	11.26	11.41	11.24	11.70	11.65	11.39	11.75

f_5 (Bohachevsky)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		16.16	16.16	16.16	16.16	15.72	15.72	15.72	15.95	15.87	15.83
g		16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16
σ		16.07	16.38	16.66	15.70	16.25	15.81	15.71	15.26	**14.26**	**12.36**
N	16.16	16.16	16.16	16.16	16.16	15.14	15.14	15.92	15.48	15.94	15.38
k_p		16.16	16.16	16.16	16.16	16.16	15.46	15.46	16.40	16.37	16.77
k_s		16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16	16.16
all		15.68	15.53	15.79	15.69	15.98	16.23	16.29	15.13	**14.04**	**12.13**

f_6 (Griewank)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		0.85	0.79	0.85	0.81	0.90	0.83	0.77	0.94	0.79	0.87
g		0.87	0.82	0.80	0.78	0.88	0.80	0.92	0.81	0.83	0.84
σ		0.80	0.78	0.88	0.80	0.80	0.79	0.76	0.81	0.82	0.87
N	0.87	0.87	0.87	0.87	0.87	0.86	0.79	0.79	0.80	0.82	1.00
k_p		0.87	0.87	0.87	0.87	0.87	0.87	0.87	0.84	0.84	0.91
k_s		0.87	0.87	0.87	0.87	0.84	0.84	0.87	0.88	0.79	0.88
all		0.92	0.85	0.80	0.90	0.78	0.85	0.83	0.86	0.82	0.91

f_7 (Shekel)

	St	0.01	0.02	0.03	0.05	0.10	0.15	0.20	0.30	0.50	0.80
μ		-1.78	-1.81	-1.66	-1.33	-1.72	-1.70	-1.43	-1.37	-1.94	-1.32
g		-1.97	-1.97	-2.00	-2.02	-2.03	-2.32	-2.32	-2.32	-1.97	-1.71
σ		-1.96	-1.96	-1.96	-1.96	-1.97	-1.97	-1.97	-1.97	-1.97	-1.97
N	-1.97	-1.97	-1.97	-1.97	-1.97	-1.98	-1.98	-1.36	-2.01	-1.32	-1.37
k_p		-1.71	-1.74	-1.66	-1.66	-1.47	-1.74	-1.71	-1.35	-1.96	-1.63
k_s		-2.00	-2.03	-2.00	-2.03	-2.03	-2.02	-2.03	-2.03	-2.06	-2.02
all		-1.84	-2.14	-1.99	-1.84	-1.78	-2.16	-1.50	-1.67	-1.53	-1.85

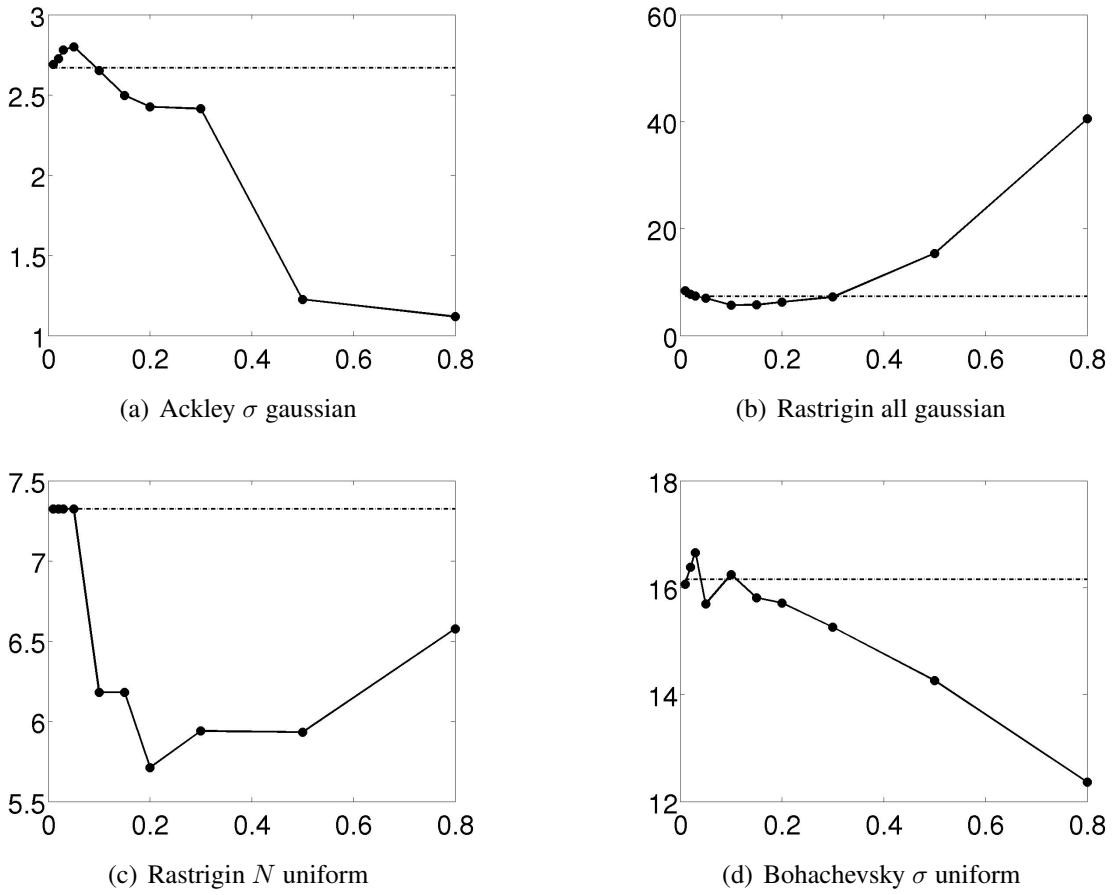
over the static values. All significance test are performed using the distribution-free two-sample Kolmogorov-Smirnov test (see Appendix B).

From these results we can see that we can achieve better performance by just varying the value of the parameter around the tuned value, without any strategy or purpose. For 4 out of 7 problems and for 9 out of 49 combinations of problem and parameter, there exists some kind of variation that leads to significantly better performance. The gaussian distribution seems to yield better results, however, there are also cases where drawing values from a uniform distribution is beneficial when compared to keeping parameters fixed. An important observation is that, in most cases where changing parameter values is beneficial, performance improves as the range of the change becomes wider, with the best results achieved when the range is 80% of the center value. Figure 5.1 compares the performances for several functions where random variation improves performance; the influence of the width of the distribution (d) for varying certain parameter values is displayed versus the obtained accuracy.

The parameter that has most often a positive response is the mutation step size σ but there are also cases where varying the population size and number of crossover points is beneficial. Finally, for function f_2 varying all parameters significantly improves performance while each parameter independently does not. This implies that, for the specific problem, the effect of the combined variation of all (or at least some) parameters is necessary for an improvement to occur.

The results of the tuning process for experiment 2 are shown in Table 5.4. In this case, the width of the distribution is subject to parameter tuning and the results only concern f_1 (Ackley). For all parameters, the tuning process converged to deviation values far from zero except for g which ends up at values relatively close to zero. Using the best vector of deviations, the SES was run 30 times. A comparison with keeping the parameters static and with results from experiment 1 is shown in Figure 5.2. Here, the two best results of experiment 1 are considered, namely the case with variation among all parameter values with a width of $d = 0.3$ and varying only σ with a width of $d = 0.8$. The tuned deviations produce better results than static and than varying all parameter values with a fixed deviation (experiment 1). However, they are not better than varying only σ. This may be due to the fact that σ is the only parameter that offers a benefit when varying or that the tuning process had trouble concurrently tuning all six deviations for all parameters.

The results of experiment 3 are presented in Table 5.5 showing the performance of the SES using static values, completely tuned variation with a gaussian and uniform distribu-



(a) Ackley σ gaussian

(b) Rastrigin all gaussian

(c) Rastrigin N uniform

(d) Bohachevsky σ uniform

Figure 5.1: *Four cases from the results of experiment 1. Each subgraph shows the performance when varying a parameter (or all) according to a random distribution. The x-axis is the width d of the distribution. The horizontal dashed line shows the performance when keeping the parameter values static to the tuned values. The caption of each subgraph lists the test function, which parameter is varied and the type of the random distribution. Lower values are better for all functions.*

Table 5.4: *Results of experiment 2 showing the best three parameter vectors found by Bonesa. For each parameter the tuned deviations are presented.*

Parameter	Vector 1	Vector 2	Vector 3
μ	17.923	19.616	25.785
g	0.025	0.06	0.03
σ	0.862	1.049	0.897
N	2.692	1.064	2.771
k_p	25.464	21.966	25.662
k_s	39.978	22.644	38.916

tion as well as a completely tuned sine function. Underlined values denote a significantly best performance. We can again see that varying the parameter values can result in better

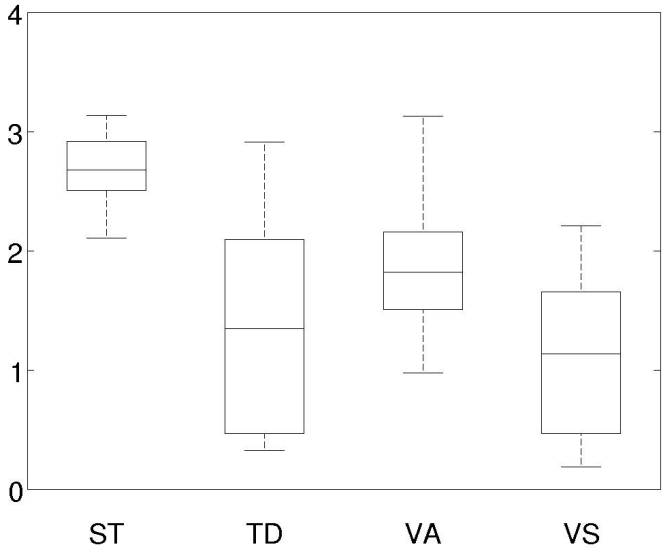


Figure 5.2: *A comparison of the performance when keeping all parameters fixed to the tuned values (ST), when varying all parameters with a gaussian distribution with a tuned standard deviation (TD), when varying all parameter with a gaussian distribution with a standard deviation with $d = 0.3$ (VA) and when varying only sigma with a gaussian distribution with $d = 0.8$. Lower values are better.*

Table 5.5: *Results of experiment 3. For each problem, the performances using the variation methods with tuned settings and the performance keeping the values fixed to the tuned values are shown. Lower values are better for all functions. For each column, underlined values denote the best and bold values denote performance not significantly worse than the best.*

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
gaussian	**1.74**	14.57	**9.58**	15.88	**14.28**	6.18	-1.26
uniform	3.08	12.17	13.28	18.34	21.76	3.54	**-2.24**
sine	3.21	48.27	38.76	14.13	15.40	17.14	**-3.08**
static	2.67	**7.32**	**7.56**	**11.51**	16.16	**0.87**	**-1.97**

performance in some cases. However, tuning the settings of the random variations (gaussian and uniform) did not produce any improvement compared to the results acquired by experiment 1 (see Tables 5.2 and 5.3). For function f_5 the performance of the tuned gaussian is much worse than the performance acquired simply by setting the deviation of all parameters to $0.8 \cdot p_i(j)$. Furthermore, for functions f_2 and f_4, while experiment 1 showed improvement when varying all parameters, here we see worse performance. It seems that the tuning process was not particularly successful in this case as it was not able to find the settings of experiment 1 or (possibly) better[3]. Nevertheless, an interesting result did arise

[3]Notice that the settings to be tuned per parameter were increased in this experiment: for the gaussian and

from this experiment. The tuned sine wave variation performs the best with problem f_7; the corresponding parameters variation is shown in Figure 5.3. Except for μ, the variation of all other parameters is a very fast oscillation within a certain range, resembling more a random sampling and less a gradual schedule.

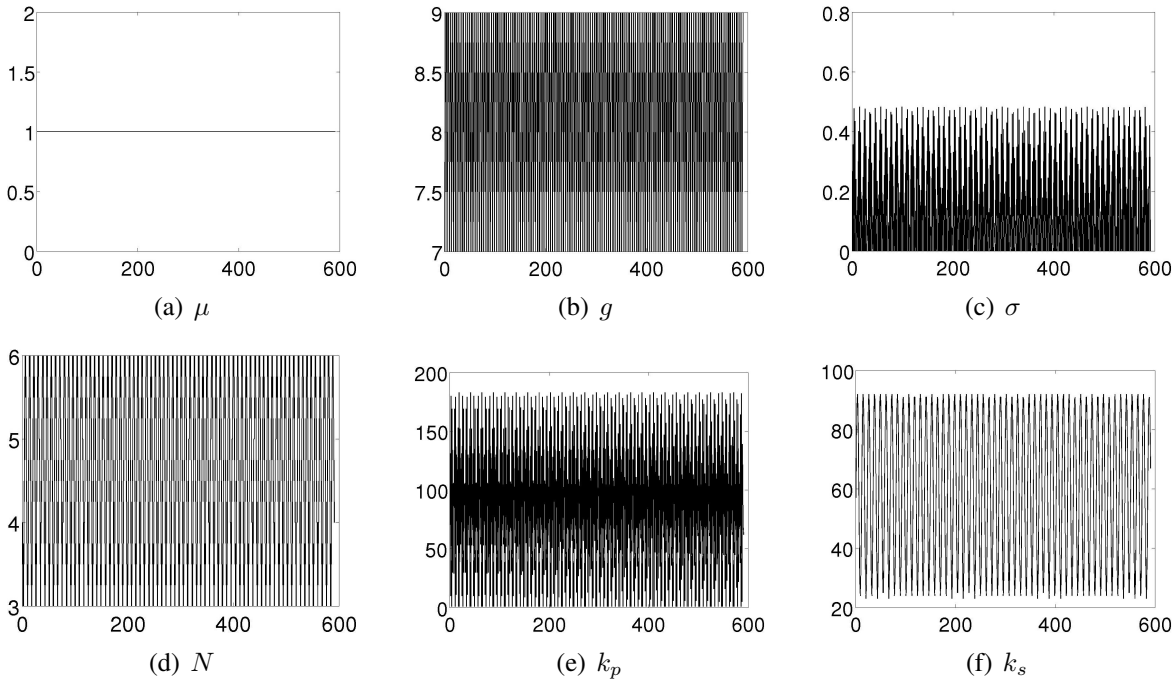


Figure 5.3: *The parameter values over time when using the sine wave with the tuned settings from experiment 3 with f_7. Each subgraph shows the values of a parameter over the generations.*

5.3 Implications and Consequences

There are some important implications following from the results of the experiments presented in this chapter. First, it is shown that there is indeed an intrinsic value in varying the values of the parameters of an EA. The fact that the variation used is mere random sampling and that the comparison is made with tuned static values and not arbitrary sub-optimal settings makes this result stronger and shows that the variation of the parameters itself can offer an advantage that is not attainable by static values, no matter how good. Furthermore, the fact that in one case there was an improvement only when all parameters were varied suggests that, for some problems, the combined control of several (or all) parameters

uniform distributions there are double the settings (two settings per parameter) while for the sine wave the factor is four.

is required in order to have some improvement in performance while controlling only one parameter may not have the same effect. This supports our view of generic control over an EA as a whole instead of arbitrarily chosen aspects of it. Of course, as the results of experiment 2 show, this brings up the question of how to effectively solve the problem of controlling several parameters concurrently and effectively since the complexity increases with the number of controlled parameters and their interactions.

Second, the results of this chapter raise an important issue in methodology. When evaluating a controller, performing a comparison to the equivalent EA with static parameter values is not enough because, as the results here show, such a comparison does not necessarily show that the controller is good. We believe that a complete evaluation of a control mechanism should also include an analysis of how the parameters are varied during a run and a comparison to "naive" variation of the same parameters as a baseline benchmark. This offers a first check into the evaluated controller's strategy and "intelligence". e.g. a repetitive time-based schedule can be seen as a control strategy (deterministic control according to [80]) but, as experiment 3 shows, it may be the case that the optimal setting of such a schedule resembles more a random sampling, meaning that it is the variation itself and the availability of multiple parameter values that improves performance and not the schedule itself.

Our conclusions here are based on experiments with only one (simple) EA; in the following chapter, these findings will be confirmed with more and more sophisticated EAs.

6

Off-the-shelf Parameter Control

THE overall classification of parameter setting and the relation of control and tuning, illustrated in Figure 2.3, shows two main categories of parameter controllers: mechanisms that need to be tuned to the EA/problem combination at hand in order to function properly and mechanisms that are robust and can be used off-the-shelf. We begin our experimental work on control algorithms in the latter category. This chapter is devoted to presenting several off-the-shelf controllers, experiments that evaluate their performance compared to benchmarks to each other and analyses on their behaviour.

The motivation for using such off-the-shelf controllers is cases when an off-line preparatory tuning process is not feasible due to time, resources or practical limitations (which has been seen in the field as a general motivation for using control instead of tuning as well). This means that they are particularly fit for one-off applications where a single instance of a specific problem needs to be solved only once (often with time restrictions and using EAs that are computation intensive and slow); in such situations, tuning is often not possible. They are also well suited for non-conventional applications with on-line, continuous and dynamic characteristics. This necessitates a control mechanism that is ready to use without requiring any further calibration; the generality of the mechanism and its ability to control

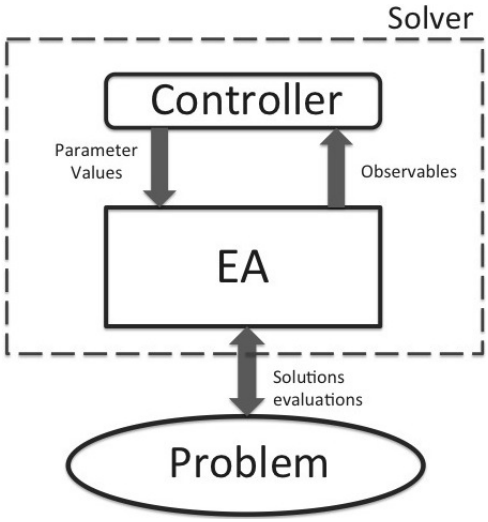


Figure 6.1: *Off-the-shelf parameter control as a plugin to the solver.*

any parameter becomes a relevant aspect here as well.

We see such a generic off-the-shelf controller as a readily usable plugin that can be added to the EA used as a solver (Figure 6.1). The workflow remains the same; the solver and controller combination work as one unit with the controller receiving observables information (see Section 2.3) and updating the EA's parameters at each iteration (generation). The only additional effort required by the designer/user is providing the controller with the values of observables and deciding the ranges within each parameter will be controlled. The latter is a relatively easy task: it is not necessary to choose "appropriate" ranges, it is up to the controller to find the good values; in that sense just using the feasible range for each parameter is a good option. On the other hand, providing observables' values requires more informed decisions and possibly some adjustments in the implementation of the EA. As we have previously pointed out (e.g. in Sections 2.3 and 3.5) the design and choice of observables is still unexplored and it is still an open question whether they are EA-dependent or controller-dependent or if there can be a selection of generally good observables. Even though we do not suggest that a controller cannot be practically implemented without observables[1], we consider feedback from the search to be an important aspect since it relates to *state* (see Section 2.3) and allows for informed control. In this thesis, we choose to address the control problem as a whole and, therefore, all the algorithms presented use observables.[2]

The following sections describe and evaluate several control algorithms based on Re-

[1] As a matter of fact several of the controllers presented in the review of Chapter 3 do not use observables.
[2] We do not, however, explore in depth the definition and influence of observables.

inforcement Learning. As explained in Section 2.4, using RL not only helps us to better understand the problem but also provides us with many existing sophisticated algorithms to adapt and apply (for an explanation of how the parameter control problem is mapped to the RL problem refer to Section 2.4). The first controller design we present in this chapter uses a discretised action space. Subsequently, algorithms with continuous actions are tested to determine if treating the continuous parameter spaces directly improves performance. The issue of defining reward is also addressed here experimentally. Finally, the most successful controller is applied to dynamic settings in order to assess one of the supporting arguments for parameter control (that it offers the necessary adaptivity in such dynamic environments).

6.1 A Discrete Controller Based on Reinforcement Learning

The first control algorithm presented in this chapter is the RL-D (Discrete RL) controller; it is based on a dynamic segmentation of the state space and an a priori discretisation of the action space. Learning is performed using a Temporal Difference approach (for a short introduction to Reinforcement Learning concepts see Appendix A). We conducted experiments using it as a 'plug-in' for several EAs that are tested on a number of different problems to answer the question whether an RL-based generic parameter controller can improve the performance of an (any) EA without the need for tailoring the controller towards the given EA/problem combination.

6.1.1 The RL-D Controller Design

First, we discuss the parameters and present a concrete list of observables and subsequently discuss the specific algorithm used for learning.

6.1.1.1 Parameters and Observables

The controller design discussed here is parameter-independent and can handle both numeric and symbolic parameters. Updates occur on every iteration (generation) of the EA.

The choice of observables is a challenging issue as was explained in Section 2.4.. So far there has been almost no research exploring the usefulness of different observables - the only exceptions we are aware of are by Veerapen et al. [278] and Whitacre et al. [285] - while

almost all controllers found in literature use fitness and/or a form of diversity as feedback from the EA without providing any motivation or justification for this choice [155]. A systematic approach for the definition of observables was suggested in Section 2.3.

Here, we choose a set of simple observables that we consider appropriate and intuitively useful (since as we have explained a better informed decision is impossible at the moment):

i. Genotypic diversity

ii. Phenotypic diversity (when different from genotypic)

iii. Fitness standard deviation

iv. Fitness improvement

v. Stagnation counter

The diversity-based observables and the fitness standard deviation are simple digests of the current EA state (no derivatives or history). They all aim at measuring different aspects of diversity which is generally accepted as an important descriptor of the state of an EA (see [155]). However, not only is the measurement of diversity dependent on the representation used by the EA but the concept of diversity is itself not well-defined [277]. Here we conduct experiments with numeric optimisation, thus we use Euclidean distance for measuring diversity, though there are alternative options (see for example [249] and [202]). The fitness improvement observable is a derivative based and is the only real fitness measure we use (defined the same as the reward that is discussed in the following section). The reason we did not include absolute fitness values is that such observables would be less useful to a non-restarting EA: states defined by absolute fitness values are not likely to occur again making anything learned from those states unusable in the future. Finally, the stagnation counter is a history-based observable; the number of generations that have passed without any improvement is a metric that might be related to taking drastic exploratory action.

6.1.1.2 Algorithm

Control actions are defined as setting each controlled parameter to a certain value. To simplify our design we cope with the continuous action space by discretising parameter ranges to equal intervals. The number of discretisation intervals is the same for all parameters, regardless of the parameter's range. When the controller picks an interval to set a parameter's value, the exact value is taken uniformly randomly from within that interval. Values

of symbolic parameters are treated each separately. Each action of the controller specifies the intervals for all numeric parameters and the values for all symbolic parameters. Consequently, the total number of control actions is equal to the product of the numbers of intervals/values of all parameters. Depending on the EA controlled, the number of parameters will change. The granularity of the discretisation is important here. The smaller the intervals, the more fine-grained the controller can set parameter values, but the larger the action space requiring more time to explore. The number of discretisation bins decides the number of actions; it is left as a choice for each specific application depending on the EA (i.e. how many parameters it has) and problem at hand (i.e. how long a run is possible).

As was discussed in Section 2.4, the choice of the reward function for an RL-based implementation of a parameter controller is not trivial because it has to be calculated after every iteration throughout the whole run but must able to approximate the goal of an optimal solution achieved at the end of the run. For our first design we chose to use the improvement of the best fitness in the current population since it is a good indicator of the progress of the search towards an optimal solution. To calculate such improvement, an obvious way would be the difference between the previous and the current fitness but that would result in disproportionate rewards: an EA tends to make large steps in the beginning even if parameters are not good while near the end very small fine-tuning improvements are very hard to achieve. This could be solved if we knew the target fitness of the problem but we cannot make this assumption in general. Consequently, we try to alleviate this problem by defining reward as the ratio of the previous fitness to the current (for a minimisation problem). Since the population and offspring sizes may be among the parameters controlled, this ratio is also normalised by the effort (number of evaluations) spent for the improvement made. Thus, we make the following definition for the reward:

$$R(s_t, a_t) = C \cdot \frac{\frac{f_b^{t+1}}{f_b^t} - 1}{Evals_{t+1} - Evals_t} \tag{6.1}$$

where f_b^i is the best fitness at time i, $Evals_i$ is the number of evaluations spent until time i and C is a scaling constant.

The core of the controller is based on Temporal Difference (TD) learning with dynamic state space segmentation and eligibility traces. The dynamic state segmentation method used is based on the work by Uther and Veloso [271]. The controller represents states as a binary decision tree. Internal nodes are decision nodes; they segment space with an

inequality on one of the observables. Leaf nodes represent actual discrete states. Each such state node S_i includes $Q(S_i, a_j)$ values and eligibility traces $e(S_i, a_j)$ for all actions a_j and a $V(S_i)$ value of the state itself. $Q(S_i, a_j)$ denotes the estimated state-action value, i.e. the expected long-term return of taking action a_j when in state S_i. These Q values are used when the controller selects actions. Eligibility traces are a way to assign credit (or blame) to actions when their influence can extend to several steps after they are taken. At each time t, every state-action pair S_i, a_j has an eligibility trace $e(S_i, a_j) \in [0, 1]$ that shows how much responsibility action a_j from state S_i is taking for the reward $R(t)$ that is presently received. The more recently action a_j was taken from state S_i the higher $e(S_i, a_j)$ is and vice versa.

At each control step the controller receives a set of observables values $\vec{o}(t)$ (as defined earlier) and a reward $R(t)$ (as defined in Eq. (6.1)). It derives the corresponding current discrete state by starting at the root of the state tree and traversing the decision nodes based on the observables values down to a state node $S(t)$. It then selects the current action $a(t)$ using an ϵ-greedy strategy: with probability ϵ a random action is chosen, otherwise it selects the action with the highest Q value: $a(t) = argmax_j Q(S(t), a_j)$. It then calculates the target value δ for the update of the previous action based on the reward $R(t)$ received. The target δ is derived according to the SARSA on-policy rule [263]:

$$\delta = R(t) + \gamma \cdot Q(S(t), a(t)) - Q(S(t-1), a(t-1)) \tag{6.2}$$

where γ is the discount rate. Before applying the δ target, the eligibility trace of the previous action is updated to one: $e(S(t-1), a(t-1)) = 1$. Subsequently, all eligible state-action pairs are updated according to target δ:

$$\begin{aligned} Q(S_i, a_j) = Q(S_i, a_j) + \alpha_t \cdot \delta \cdot e(S_i, a_j), \\ \forall S_i, a_j : e(S_i, a_j) > e_{min} \\ \alpha_t = \begin{cases} \alpha_0 & R(t) = 0 \\ \alpha & otherwise \end{cases} \end{aligned} \tag{6.3}$$

where e_{min} is the minimum eligibility and α is the step-size parameter. The value of α decides how much influence the target δ will have on the current Q values. Because rewards (fitness improvements) tend to be zero a lot of the time, we use a different (and much lower) α_0 to avoid Q values quickly dropping to zeros. The value of the state is updated to be the

maximum action value within that state:

$$V(S_i) = maxQ(S_i, a_j) \tag{6.4}$$

Finally, the eligibility traces of all state-action pairs are updated:

$$e(S_i, a_j) = e(S_i, a_j) \cdot \gamma \cdot \lambda \tag{6.5}$$

where γ is the discount rate and λ is the trace decay parameter.

After the next action is selected and all updates are performed the state tree is updated. The state tree starts with only one root node which is a state node representing one universal state; any possible observables $\vec{o}$ will map to that state. Every time a control step is made a transition occurs from an observation and action to a reward and a new observation $T_t = (\vec{o}(t), a(t), R(t+1), \vec{o}(t+1))$. Every state maintains an archive of such transitions and at each control step the transition T_t is added to the archive of state $S(t)$. If the state's archive is larger than a threshold A_m and with probability p_s the state is checked for splitting into two new states, which means that the current node becomes an internal decision node and two new state nodes are added as its children. To convert the state node into a decision node a choice of observable and a splitting value are required to form the corresponding inequality.

The purpose of this process is to divide a state into two parts with significantly different value estimates. First, each transition in the state's archive is assigned a value equal to the reward of the transition plus the current value estimation of the state that currently corresponds to the *end* observation of the transition (the $\vec{o}(t+1)$ for T_t). Subsequently, all observables are checked as candidates for the splitting criterion. For each observable:

- Transitions in the archive are sorted according to the current observable in the transition's *starting* observation (the $\vec{o}(t)$ for T_t).
- Taking the sorted list of transitions we consider the values of the current observable. The mid-point between the observable values of every two subsequent transitions in the sorted list is a candidate for a splitting point[3]. Splitting points are only considered if they split a transition list to fractions that are larger than a A_f.
- Given the split point, the values of the transitions of the two parts form two samples. A Kolmogorov-Smirnov double test is run with these two samples and the resulting

[3]If the transitions are too many then only 100 equally spaced points are checked

D value of the test is saved.

– The above process is repeated for all observables.

After all observables are checked, the smallest D value is taken. If it is smaller than D_{max} then the node is split at the corresponding observable and splitting point. Two new state nodes are created and their Q, V and e values are set to the values of the parent node as an initial estimation. The archive of the parent node is split according to the chosen split point and the parts are given to the corresponding children nodes. The parent node becomes a decision node with an inequality using the observable and split point derived above.

The RL controller has ten meta-parameters due to the TD learning rule, the dynamic state tree and the eligibility traces. This number of meta-parameters can be seen as too high when considered from a quid-pro-quo point of view; in that sense trading the EA parameters for (maybe even more) meta-parameters may seem futile. However, we argue that the meta-parameters of the controller are justifiable for two main reasons: (i) as was mentioned in Section 2.2, one of the arguments in favour of parameter control is that the meta-parameters of controllers are less influential than EA parameters and much easier to deal with, and, most importantly, (ii) the controller does not merely replace EA parameters with its own meta-parameters but adds the value of the dynamic variation of EA parameters. Here, we have set the meta-parameters of RL-D to default values according to literature or simply intuition. These default values are used for RL-D in all the experiments of this thesis. The meta-parameters of RL-D and their default values are shown in Table 6.1.

Table 6.1: *RL-D controller meta-parameters.*

TD		State tree		Traces	
Parameter	Value	Parameter	Value	Parameter	Value
ϵ	0.1	A_m	60	λ	0.8
γ	0.8	A_f	0.2	e_{min}	0.001
α	0.9	D_{max}	0.05		
α_0	0.2	p_s	0.1		

6.1.2 Experimental Setup

The experiments presented in this section aim at evaluating the RL-D controller described in the previous section. To this end, we selected four different EAs (described later) and four different parameter control mechanisms: none (using the default parameter values of

the given EA), PRAM [289], RL-D, and random variation of parameter values. In order to get results as realistic and meaningful as possible, each EA was tested with functions it was designed for or is considered competent solving. For the details of the EAs (and their parameters), the test problems and the controller benchmarks refer to Appendix B.

6.1.2.1 EAs and Problems

We decided to use algorithms and problems from the classic numeric optimisation domain which do originate from real world applications but are also widely used and understood in the research community. This also allows us to study a wide range of problems as well as the impact of the level of sophistication of the algorithm upon the performance of the RL controller. For our experiments we selected four different EAs ranging in sophistication from very simple to state-of-the-art. When selecting the problems for the EAs there are two options. We could use the same problems for all EAs or specify a different set for each of them separately. In our opinion, the second option is preferable because the EAs have been designed for and validated on different type of problems. Using targeted problems makes the task for the controller more challenging, thus, the study more interesting. In particular we used:

i. An unspecialised "naive" Simple ES (SES) solving four standard numeric optimisation test functions frequently used in literature: Rastrigin, Schwefel, Schaffer and Fletcher & Powell (all in 10 dimensions). Some of the SES parameters were fixed for this experiment to reduce the complexity of the control problem: the crossover used was uniform, survivor selection was "plus" and the parent tournament size was fixed to 2.

ii. The Cellular GA (CGA) solving BBOB functions 21 through 24 in 40 dimensions (we chose the hardest functions of the noiseless BBOB testbed that would justify long runs and resemble the real functions tackled in one-off applications).

iii. The GA-MPC tested with the test suite it was designed for (CEC2011); we used two problems it performed very well (problems 12 and 13) and two for which it performed poorly (problems 7 and 11.8).

iv. The IPOP-10DDr CMA-ES solving BBOB functions 21 through 24 in 40 dimensions (we chose these functions for the same reason as for the Cellular GA).

The default parameters values of the EAs are listed in Appendix B. The ranges of control are shown in Table 6.2.[4]

6.1.2.2 Control mechanisms

Each algorithm was run in combination with four different control mechanisms. The baseline mechanism was the use of no control, that is, using static parameters set to default values as provided in the source codes or specified in the relevant publications of the EAs listed above. (For our own SES we set parameters to "standard" values).

All algorithms were also run with the RL-D controller. The parameters of the RL-D controller were set to the default values shown in Table 6.1 for all experiments. The only option of the RL-D controller changed was the number of discretisation bins: since the number of parameters and the length of runs differ among EA and problem settings, the amount of discretisation bins was chosen for each experiment separately to yield a reasonable number of control actions.

Third, we run all algorithms with one of the few existing generic controllers, the Probabilistic Rule-driven Adaptive Model (PRAM) by Wong et al. [289]. PRAM can only handle numeric parameters, thus, any symbolic parameters were kept static to their default values when running with PRAM. For all PRAM experiments the epoch length was set to 150, 20% of which was the training phase. PRAM's step sizes were separately adjusted for each algorithm and parameter since parameter ranges differ drastically.

Finally, we also run all algorithms with their parameters being randomly varied (as was motivated in Chapter 5). In these tests the ranges of parameters were the same as for the RL-D controller but values were drawn uniformly randomly.

In this setting, we are interested in achieving maximal performance given a certain amount of effort, thus, the termination criterion of all runs was reaching the maximum number of evaluations ($15 \cdot 10^4$ for all runs with the GA-MPC and 10^6 for all other runs). Each algorithm/problem/control mode combination was run 30 times. Table 6.2 summarises the experimental setup showing all algorithms, their parameters and their control ranges, the test problems for each algorithm, the number of action discretisation bins used for the RL-D controller and the step sizes of the PRAM.

[4]The ranges of control may differ from the domains listed in Appendix B, e.g. for parameters with open infinite theoretical domains, we chose control ranges that we consider to be reasonable.

Table 6.2: *Experimental Setup*

EA	Parameters, ranges	Problems	RL-D settings	PRAM settings
Simple ES	$\mu \in [10, 80]$ $g \in (0, 7]$ $\sigma \in (0, 2]$ $k_s \in [2, 80]$	Rastrigin, Schwefel, Schaffer, Fletcher&Powell in 10 dimensions	Action bins: 5	$s(\mu) = 8$ $s(g) = 0.25$ $s(\sigma) = 0.1$ $s(k_s) = 3$
Cellular GA	$xover \in \{2p, Ar\}$ $p_c \in [0.6, 1]$ $mut \in \{G, U, G_d, C\}$ $p_m \in (0, 0.4]$ $m_{var} \in (0, 0.4]$	BBOB2013 f_{21}, f_{22}, f_{23}, f_{24} in 40 dimensions	Action bins: 4	$s(p_c) = 0.04$ $s(p_m) = 0.04$ $s(m_{var}) = 0.04$
GA-MPC	$\mu \in [50, 130]$ $k_{max} \in [3, 15]$ $\beta_m \in [0.3, 1.1]$ $\beta_{std} \in (0, 0.5]$ $p_r \in (0, 0.4]$ $f_a \in [0.3, 0.7]$	CEC2011 f_7, $f_{11.8}$, f_{21}, f_{22}	Action bins: 4	$s(\mu) = 6$ $s(k_{max}) = 1$ $s(\beta_m) = 0.05$ $s(\beta_{std}) = 0.05$ $s(p) = 0.04$ $s(f_a) = 0.04$
IPOP-10DDr CMA ES	$c_c \in [0, 1]$ $c_s \in [0, 1]$ $d \in [1, 5]$ $\mu_{cov} \in [1, 20]$ $xover \in \{E, D_l, D_{sl}\}$	BBOB2013 f_{21}, f_{22}, f_{23}, f_{24} in 40 dimensions	Action bins: 4	$s(c_c) = 0.1$ $s(c_s) = 0.1$ $s(d) = 0.5$ $s(\mu_{cov}) = 2$

6.1.3 Results and Analysis

The results of all experiments are shown in Table 6.3. The RL-D controller significantly improves over the static mode in 8 out of 16 cases while it has a better best run in additional 4 cases. For PRAM these number are 7 and 3 and for random 9 and 3. The RL-D controller has one less success case (better than static) than random but, in 5 of the common success cases, the RL-D controller is significantly better than random. All significance tests are performed using the two-sample Kolmogorov-Smirnov test (see Appendix B).

In several cases, the performance of the EA is improved by the RL-D controller compared to the static mode while there are only two cases where performance is significantly worse with RL-D control. Furthermore, as we stated in the previous section, the settings of RL-D were the same across all experiments (with the trivial exception of the number of action bins). Based on these results we can conclude that a generic RL-based controller can indeed improve the performance of an EA with no need of tailoring to the given problem.

Table 6.3: *Results of all experiments evaluating RL-D. There is a section for each algorithm showing the performance of the algorithm with four parameter modes: static default, randomly varied, controlled by PRAM and controlled by RL-D. Performance is shown as final fitness averaged over all repeats (ABF) and the final fitness of the best run. In the ABF columns underlined values are significantly better than static and bold values denote the winner(s) (not significantly worse than the best). Significance was decided with a two-sided Kolmogorov-Smirnov test with $\alpha = 0.05$. All problems are minimisation.*

Simple ES

	-Rastrigin-		-Schwefel-		-Schaffer-		-FP-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	27.68	20.72	817.53	236.98	22.04	6.93	7935.47	1982.58
random	4.65	1.63	**552.13**	118.44	**3.07**	1.98	420.26	187.94
PRAM	11.76	2.18	**422.49**	0.00	4.25	0.91	**1022.72**	15.56
RL-D	**0.67***	0.04	779.27	0.00	9.85	1.08	**166.21***	3.42

Cellular GA

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	40.89	40.81	-998.83	-999.09	9.06	8.57	**471.81***	435.15
random	40.84	40.80	-998.92	-999.12	8.64	8.30	490.17	422.01
PRAM	**40.78***	40.78	**-999.30***	-999.49	9.28	8.44	493.93	412.91
RL-D	40.80*	40.78	-999.18*	-999.26	**8.49***	8.15	**484.62**	412.93

GA-MPC

	-CEC2011 f_{7}-		-CEC2011 $f_{11.8}$-		-CEC2011 f_{12}-		-CEC2011 f_{13}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	1.68	0.84	1945573	1510399	**16.24**	14.11	**15.25**	8.61
random	**0.90**	0.58	**946705**	939736	**16.54**	13.72	**17.63**	8.79
PRAM	1.55	0.59	1580994	941087	**16.20**	14.23	17.43	8.86
RL-D	**0.96**	0.63	**947905**	941267	**16.42**	12.59	19.38	14.12

IPOP-10DDr

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**41.39**	40.78	**-997.67**	-999.31	**6.89**	6.87	**131.54**	108.03
random	**41.75**	40.78	**-997.88**	-999.31	**6.94**	6.87	**126.46**	107.75
PRAM	41.99	40.78	**-997.63**	-999.31	6.91	6.87	**137.57**	109.71
RL-D	**41.69**	40.78	**-997.75**	-999.31	6.90	6.88	**128.91**	106.93

Looking at the results of Table 6.3 we can see that the RL-D controller has a much better effect for some EAs (notably the SES) while it is not particularly advantageous for others such as the CMA-ES. One way to interpret this is by considering the margin of improvement for each algorithm when solving a specific problem, i.e. how well tailored is the algorithm to the problem at hand. The SES is very crude while the Cellular GA, though more complex, is certainly not a competent numeric optimiser. Neither of them is particularly fit for the problems they are solving. On the other hand, the GA-MPC is an efficient optimiser (winner of

the CEC 2011 competition) but notice that it ranked low for the first two problems while it ranked first and second for the last two problems. Finally, the CMA-ES is considered the most competent numeric optimiser and consistently performs well for the BBOB competitions. Table 6.4 summarises[5] these observations along with a simple comparison between the static mode and the RL-D control. It shows that when there is margin for improvement for the specific EA/problem combination, the RL-D controller will exploit it.

Table 6.4: *RL-D Control vs static: ++ (or --) shows that the ABF is significantly better (or worse) while + (or -) shows that ABF is not significantly different but the result of the best run is better (or worse). A 0 means no difference. Grey cells denote that the EA is particularly fit to the specific problem (see explanation in the text).*

SES	++	+	++	++
CGA	++	++	++	+
GA-MPC	++	++	+	--
CMA ES	0	0	--	+

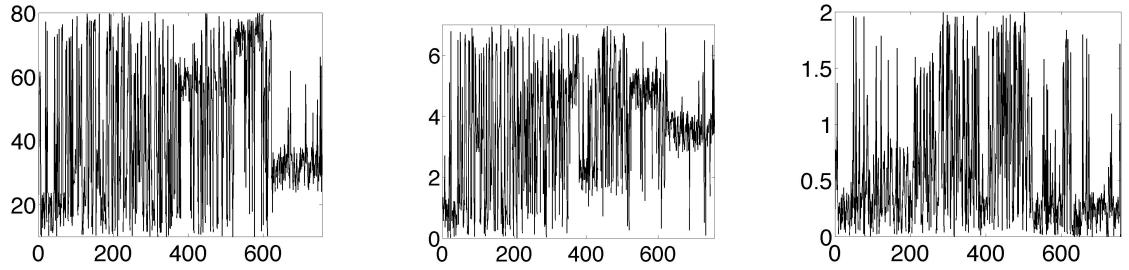
Another factor that may contribute to the performance differences among EAs when using the RL-D controller is monotonicity (decided by restarting and elitism). The SES is non-elitist, the Cellular GA and GA-MPC are elitist and do not restart while the IPOP-10DDr CMA-ES typically restarted only four or five times per run. This is relevant if we look into the observables' values. It is frequently seen that, during a run, observables will follow a monotonous increasing or decreasing curve (unless the EA restarts or is non-elitist). In such cases specific observations only occur once, thus, states do not repeat unless the EA is restarted[6]. Consequently, elitism combined with no restarting could mean that anything learned during a run is never actually used and the controller ends up solving a dynamic multi-armed bandit problem (i.e. state is no longer useful).

Looking at the parameter values set by the RL-D controller we could not discern any obvious pattern. Even when looking at one of the best performing runs for RL-D (SES with the Rastrigin problem) in Figure 6.2 no apparent trends can be seen. However, that is not necessarily a problem considering the combined effect of multiple observables, the dynamic segmentation of the state space and the effects of exploratory actions.

For all experiments, the settings of the RL-D controller were fixed to the default values shown in Table 6.1 without making any further adjustments or tests. Though that is not

[5]Directly derived from the raw data in Table 6.3.

[6]Non-elitism can result in "weak" restarting with the diversity and fitness hopping to large/small values every time the best individuals do not survive.



(a) μ (b) g (c) σ

Figure 6.2: *Parameter values over time for the best run of the SES solving the Rastrigin function with RL-D control.*

enough to derive conclusions about the robustness of the RL-D controller, it shows that any improvements in performance reported in this paper were achieved with no additional effort for applying the RL-D controller to the EAs.

Looking at the performance of the PRAM benchmark, it has a slightly worse success (better than static) count than RL-D but there are two EA/problem combinations where PRAM outperforms all other parameter modes. A noteworthy observation is that random variation of parameter values had, in many cases, a significantly positive effect on performance, even for the more complex GA-MPC. This confirms the results of Section 5 and extends them to more and more relevant EAs.

6.1.4 Discussion

This section presented and evaluated the RL-D design for a generic parameter controller. Experimental results showed that it can enhance performance without the need for tweaking its parameters. A detailed analysis shows that the RL-D controller can exploit an existing margin of improvement, i.e. when the EA has not been fully tailored towards the particular problem at hand. Compared to the PRAM and random variation benchmark controllers, the RL-D controller generally outperforms PRAM but not random (at least not in terms of improving over static). An analysis of values selected by the controller shows that, though the controller is able to improve performance, a pattern in the selection of parameter values is hard to discern. A surprising - but in retrospect logical - observation is that the EA's monotonicity can have an influence on the efficiency of the RL-D controller. Finally, another interesting observation is that random variation of parameter values can enhance the performance compared to static default values.

Two decisions made in the design of RL-D – namely the simplifying a priori discretisation of the action space and the choice of the reward definition – will be reexamined in the following chapters.

6.2 Control with a Continuous Action Space

The design of the RL-D controller presented in the previous section included a simplifying choice: parameter ranges were a priori discretised into equal intervals in order to allow for an algorithm that handles a discrete action space. However, this a priori discretised action space may be problematic if the "sweet spots" of parameters are smaller than the intervals used or split by them. Furthermore, the random choice of values within intervals creates noise while a finer discretisation would yield too many actions to explore. This simplifying discretisation choice was made because reinforcement learning techniques are primarily applied to discrete problems with the continuous environments requiring specialised (and more sophisticated) approaches usually involving some function approximation to estimate the value function or the policy (or both)[273]. Santamaria et al. [240] compared different alternatives of function approximation for the action value function, van Hasselt and Wiering [274] employed neural networks both for estimating the value function and representing the policy and Melo and Lopes [204] used regression for the value function and natural gradient updates for the parameterised policy. Methods that use other techniques also exist: Pazis and Lagoudakis [221] used state decomposition to transform the problem, Lasaric et al. [176] suggested sequential Monte Carlo to approximate the optimal policy and Milln et al. [207] used the weighted average of discrete actions.

In this section we modify the discrete design of RL-D and use two approaches (based on interpolation of fixed points and sampling) to directly treat the parameter space as a continuous action space. This deals with the issues mentioned above regarding the arbitrary segmentation of parameter ranges and the noise created by random choices within intervals. We evaluate these continuous approaches by testing the continuous controllers with the same EAs and problems used in the previous section and comparing them to RL-D as well as the same benchmarks, i.e. PRAM, random variation and static default parameters. With these experiments we attempt to answer the question whether a better treatment of continuous parameter values can improve performance of the off-the-shelf RL-based parameter controller.

Finally, we experiment with meta-evolution (ME) as a control method. Meta-evolution [102] has been used by several authors for finding good static values of parameters. Grefenstette [116] used meta-GAs to find good static values for the parameters of a GA solving numeric problems, Back [20] suggested a parallel meta-ES for tuning a GA, Freisleben and [103] followed a more structured approach by also evolving operators (e.g. selection type) of the problem solving GA. Meta-evolution can be seen as a promising option for a generic, out-of-the-box controller that can easily handle any parameter (both numeric and symbolic). However, there is currently no straightforward way for the meta-EA to make use of the observables describing the state of the solving EA. Thus, a ME controller can be seen as a "blind" (no use of observables) controller with a dynamic mechanism.

6.2.1 The Continuous Controllers' Design

As discussed earlier, the simple a priori discretisation of parameter ranges poses some potential problems. Here, we employ two more sophisticated RL methods for treating the continuous action space of the controllers. The first method is based on interpolation of fixed points and the second on a sample of points. Both these continuous methods only replace the action discretisation; the other parts of the controllers' designs are the same as in RL-D (parameters, observables, dynamic state space segmentation). The new methods for the continuous action space are described in the following two subsections.

6.2.1.1 RL-IP: Interpolating fixed points

This approach is based on Milln et al [207] and involves interpolating neighbourhoods of fixed points in the action space. This allows continuous actions as the weighted average of several points and a constant exploration of the area around the point currently known as best. We use a set of K fixed values $\{v_1^x, v_2^x, ..., v_K^x\}$ in equally spaced intervals of length L_x in the range of each parameter x (for symbolic parameters all values are taken). The product of all these values yields a grid of K^N points $(v_i^1, v_j^2, ..., v_z^N)$, $i, j, ..., z \in [1, K]$ (for simplicity assume that symbolic parameters have K values) in the action space; each point p is a fixed action. Action values $Q(p, S)$ are maintained for every point p and state S.

To derive the continuous action a for a state S, the centre point p_c with the highest value is taken, $p_c = argmax_p(Q(p, S))$. The neighbours P_n of p_c are all the points that are within

a "step" (for all dimensions) from the centre point in the grid:

$$P_n(p_c) = \{p | \forall x : |v^x(p) - v^x(p_c)| \leq 1\} \tag{6.6}$$

where $v^x(p)$ is the index of the value of p in dimension x. All neighbours $p_i^n \in P_n$ are assigned a weight proportional to their Q value:

$$w_i = \frac{1}{|P_n| + (Q(p_c, S) - Q(p_i^n, S))^2} \tag{6.7}$$

The action a is the weighted average of the centre point and its neighbours:

$$\begin{aligned} a(x) = p_c(x) + \sum_{i=1}^{|P_n|} w_i \cdot (v^x(p_i^n) - v^x(p_c)) \cdot L_x, \\ for\ numeric\ parameters\ x \\ a(x) = p_c(x),\ for\ symbolic\ parameters\ x \end{aligned} \tag{6.8}$$

Similarly, the value of a is

$$Q_a = Q(p_c, S) + \sum_{i=1}^{|P_n|} w_i \cdot Q(p_i^n, S) \tag{6.9}$$

When the reward of action a is received, eligibility traces of the points used for a (meant to credit or blame a point for the reward) are updated as

$$\begin{aligned} e^t(S, p_c) &= e^{t-1}(S, p_c) + \frac{1}{1 + \sum_{i=1}^{|P_n|} w_i} \\ e^t(S, p_j^n) &= e^{t-1}(S, p_j^n) + \frac{w_j}{1 + \sum_{i=1}^{|P_n|} w_i} \end{aligned} \tag{6.10}$$

The rest of the update is the same as with the RL-D controller, using Q_a to calculate δ for the SARSA rule. For exploration, an ϵ-greedy approach is also used: with probability ϵ a random point is picked and noise is added to every numeric parameter x with values from $N(0, \frac{L_x}{5})$.

6.2.1.2 RL-SMC: Sample approximating probability distribution

This method is based on Lazaric et al. [176]. Each state maintains a sample of N points in the action space. This sample is initialised uniformly randomly with equal weights for the starting global state. Each sample point has a Q value, a weight and an eligibility trace.

Actions are selected probabilistically: a sample i is picked with probability

$$p_i = \frac{w_i}{\sum_j w_j} \tag{6.11}$$

The TD update is the same as in 6.1.1, using the Q and trace values of the last applied point. After the Q values are adjusted and normalised, the weights are updated using a Boltzmann exploration strategy

$$w_i^t = w_i^{t-1} \cdot \frac{e^{\frac{Q^t(p_i,S)-Q^{t-1}(p_i,S)}{\tau_S^t}}}{\sum_j w_j^{t-1} \cdot e^{\frac{Q^t(p_j,S)-Q^{t-1}(p_j,S)}{\tau_S^t}}} \tag{6.12}$$

This updates the probability distribution of the current sample according to the current estimation of the action-value function (the Q values). The parameter τ_S is the temperature defining the degree of exploration and it decreases over time: $\tau_S^t = \frac{\tau_{init}}{1+\tau_\delta \cdot t_S}$. Each state S has its own τ_S because states are created at different times and, thus, have different "ages" t.

As the probability distribution changes to approximate the optimal policy, certain weights will become too small and a resampling step is required to replace action points with low weights with new points in the "high-weight" areas. Resampling is triggered for a state when its effective sample size $N_{eff} = \frac{1}{\sum_i w_i}$ becomes smaller than σ. To avoid too fast convergence, a $\frac{\tau_S}{2 \cdot \tau_{init}}$ fraction of the sample is taken uniformly randomly from the previous while the rest are chosen using the probabilities p_i as when selecting an action. To avoid filling the whole sample with just copies of a few points a smoothing step follows. Each point in the new sample is updated as

$$p(x) = U[\frac{p_l(x) + p(x)}{2}, \frac{p_r(x) + p(x)}{2}] \tag{6.13}$$

where p_l and p_r are the closest neighbours of p *in dimension* x so that $p_l(x) < p(x)$ and $p_r(x) > p(x)$.

When a state is split, the new states inherit the parent's sample (but their τ values start fresh allowing for new exploration). The rest of the dynamic state space handling is the same as with RL-D.

6.2.2 Experimental Setup

This experiment is primarily aimed at investigating whether the continuous handling of the parameter space can improve the performance of the RL controller. It compares the discrete RL-D controller, the continuous RL-IP and RL-SMC controllers, a meta-evolutionary approach, PRAM, random variation and default static values. For the details of the EAs, problems and controller benchmarks used, refer to Appendix B.

6.2.2.1 EAs and Problems

The EA and problem setup is similar as in the previous section (see 6.1.2):

i. The Simple ES (SES) solving four standard numeric optimisation test functions frequently used in literature: Rastrigin, Schaffer, Schwefel and Fletcher & Powell (all in 10 dimensions).

ii. The Cellular GA (CGA) solving BBOB functions 21 through 24 in 40 dimensions (we chose the harder functions that would justify long runs and resemble the real functions tackled in one-off applications).

iii. The GA-MPC tested with the test suite it was designed for (CEC2011); we used two problems it performed very well and two for which it performed poorly (12/13 and 7/11.8 respectively).

iv. The IPOP-10DDr CMA-ES solving BBOB functions 21 through 24 in 40 dimensions (we chose these functions for the same reason as for the Cellular GA).

6.2.2.2 Control mechanisms

We tested seven different parameter control mechanisms:

i. None, i.e. keeping static the default parameter values of the EA as found in the source codes or in the relevant publications of the EAs (see Appendix B).

ii. Random variation of parameter values (for a motivation see Chapter 5). At each generation, parameter values are taken uniformly randomly from their whole range.

iii. The Probabilistic Rule-driven Adaptive Model (PRAM) [289], one of the few existing generic controllers (for details see Appendix B). PRAM can only handle numeric

Table 6.5: *RL Controllers' Parameters. The TD, state tree and traces parameters are used by all three versions while the SMC column is specific to the RL-SMC controller only.*

TD		State tree		Traces		SMC	
Param.	Value	Param.	Value	Param.	Value	Param.	Value
ϵ	0.1	A_m	60	λ	0.8	τ_i	1
γ	0.8	A_f	0.2	e_{min}	0.001	τ_δ	0.005
α	0.9	D_{max}	0.05			σ	0.95
α_0	0.2	p_s	0.1				

parameters, thus, any symbolic parameters were kept static to default values when ran with it. PRAM's epoch length was set to 150 (20% exploration phase) while its step sizes were adjusted for each algorithm and parameter since ranges differ drastically. This controller is "blind": it does not use observables.

iv. Meta-evolution (ME), implemented with a CMA-ES. The search space of the CMA-ES is the parameter space of the controlled EA and each (meta-)individual of the CMA-ES is a parameter vector of the controlled EA. Every meta-individual (parameter vector) created by the CMA-ES is used for one generation by the controlled EA and the improvement in best fitness of the controlled EA is assigned as the fitness of the meta-individual of the CMA-ES. All settings and parameters of the CMA-ES were kept as default. This controller is also "blind": it does not use observables.

v. The discrete RL-D controller from Section 6.1 that discretises parameters into fixed intervals. The parameters of the RL controller were set to the values shown in Table 6.5 for all experiments. The number of discretisation bins was chosen for each experiment separately to yield a reasonable number of actions.

vi. The RL-IP controller that uses interpolation of fixed points in the parameter space as described earlier. The parameters of the RL-IP are the same as the RL-D controller. The number of points per parameter (resolution) was set for each EA for the same reasons as setting the bins for the RL-D.

vii. The RL-SMC controller that uses a sample of points as described earlier. The SMC mechanism has three additional settings (see Table 6.5) compared to the other two RL controllers. For all RL-SMC experiments, the sample size was 40.

In this setting, we are interested in achieving maximal performance given a certain amount of effort, thus, the termination criterion of all runs was reaching the maximum num-

ber of evaluations ($15 \cdot 10^4$ for all runs with the GA-MPC and 10^6 for all other runs). Each algorithm/problem/controller combination was run 30 times. For the RL-IP controller we used a resolution of 5 when controlling the SES and a resolution of 4 for the other EAs. For the control ranges of each parameter of all EAs and for the settings of the RL-D and the PRAM controllers refer to Section 6.1.2 and Table 6.2.

6.2.3 Results and Analysis

The results of all experiments are shown in Table 6.6; all significance tests were performed using the two-sample Kolmogorov-Smirnov test (see Appendix B). The observation from Section 6.1 that parameter control in general can have a significant positive impact on performance is supported with the new controllers as well, however, that improvement is not consistent over all EAs and problems. Again, for a better interpretation of the results we consider how sophisticated is each EA or how tailored it is to the problems it is solving (see EAs' ordering in the previous section)[7]. From that perspective, we again see that control (especially the RL controllers) is more effective when applied to EAs that are less sophisticated or not tailored to the problem.

A more meaningful summary of the results is derived if we consider the target area of the controllers examined: one-off problems where a control method is used out-of-the box for a single (or few) runs in order to improve the best solution the EA can find. Thus, the probability of such improvement is an appropriate indicator of the general usefulness of the controller. We get an estimation of these probabilities by measuring success counts[8] as presented in Table 6.7. The first two rows show that the methods with the highest success count are RL-D, RL-IP and random variation. The latter has the highest success count while RL-D and RL-IP equally often improve performance compared to having static default parameter values. The last two rows show that the discrete RL-D controller is the one most often better than random variation. As was explained in Section 6.1, even though random variation improves over static one time more often than RL-D, when directly compared, the RL-D controller has a significantly better ABF than random in almost one third of the test cases and a better best run for more than half.

Looking at Tables 6.6 and 6.7 we see that the RL controllers using continuous action methods are not better than the discrete RL-D even though most of the parameters controlled

[7] Also notice that GA-MPC ranked low for f_7 and $f_{11.8}$ but very high for f_{22} and f_{21}.
[8] These counts are all directly derived from the raw numbers in Table 6.6.

Table 6.6: *Results of the experiments evaluating the RL-IP and RL-SMC controllers. There is a section for each EA including subtables with the results for each test problem. Subtables show the performance of the EA with that problem in terms of final fitness averaged over all repeats (ABF) and final fitness of the best runs. In the ABF columns, underlined values are significantly better than static, bold values denote the winner(s) (not significantly worse than the best) and asterisks mark values significantly better than random. Significance is decided by Kolmogorov-Smirnov tests with* $\alpha = 0.05$*. All problems are minimisation.*

Simple ES

	-Rastrigin-		-Schwefel-		-Schaffer-		-FP-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	27.68	20.72	817.53	236.98	22.04	6.93	7935.47	1982.58
random	4.65	1.63	**552.13**	118.44	3.07	1.98	420.26	187.94
PRAM	11.76	2.18	**422.49**	0.00	4.25	0.91	**1022.72**	15.56
ME	**0.12***	0.00	**587.15**	118.44	**1.45***	0.38	**79.87***	10.62
RL-D	0.67*	0.04	779.27	0.00	9.85	1.08	**166.21***	3.42
RL-IP	2.27*	0.00	708.24	0.00	4.47	0.70	529.67	70.05
RL-SMC	8.08	0.06	**488.34**	118.44	7.12	1.68	1027.30	16.70

Cellular GA

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	40.89	40.81	-998.83	-999.09	9.06	8.57	**471.81***	435.15
random	40.84	40.80	-998.92	-999.12	8.64	8.30	490.17	422.01
PRAM	**40.78***	40.78	**-999.30***	-999.49	9.28	8.44	493.93	412.91
ME	41.00	40.87	-998.44	-998.89	8.93	8.41	510.60	465.43
RL-D	40.80*	40.78	-999.18*	-999.26	**8.49***	8.15	**484.62**	412.93
RL-IP	40.81*	40.78	-999.05*	-999.26	8.60	7.95	510.29	397.63
RL-SMC	41.62	40.78	-998.62	-999.54	8.83	7.83	498.95	403.14

GA-MPC

	-CEC2011 f_{7}-		-CEC2011 $f_{11.8}$-		-CEC2011 f_{12}-		-CEC2011 f_{13}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	1.68	0.84	1945573	1510399	**16.24**	14.11	**15.25**	8.61
random	**0.90**	0.58	**946705**	939736	**16.54**	13.72	**17.63**	8.79
PRAM	1.55	0.59	1580994	941087	**16.20**	14.23	17.43	8.86
ME	**0.99**	0.77	**951570**	939398	17.86	13.54	19.63	14.07
RL-D	**0.96**	0.63	**947905**	941267	**16.42**	12.59	19.38	14.12
RL-IP	**0.93**	0.62	**997692**	940666	17.66	13.68	18.73	8.69
RL-SMC	**0.90**	0.63	1009638	945698	**16.95**	10.46	**17.83**	11.05

IPOP-10DDr

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**41.39**	40.78	**-997.67**	-999.31	**6.89**	6.87	**131.54**	108.03
random	**41.75**	40.78	**-997.88**	-999.31	**6.94**	6.87	**126.46**	107.75
PRAM	41.99	40.78	**-997.63**	-999.31	6.91	6.87	**137.57**	109.71
ME	**41.17**	40.78	**-997.03**	-999.31	6.92	6.87	**127.81**	106.42
RL-D	**41.69**	40.78	**-997.75**	-999.31	6.90	6.88	**128.91**	106.93
RL-IP	**41.45**	40.78	**-998.17**	-999.31	6.93*	6.88	**126.88**	106.30
RL-SMC	**41.45**	40.78	**-998.17**	-999.31	**6.90**	6.87	**126.82**	107.40

are continuous. The success ratios of RL-IP and RL-SMC are not better than that of RL-D (see Table 6.7). Furthermore, as Table 6.8 shows, there is no consistent performance gain when using the continuous RL-IP and RL-SMC versions over the discrete RL-D controller. There are only few cases where the two continuous extensions produce better results and there is no obvious relation between performance gain and characteristics of the EA and/or problem.

Table 6.7: *Summary of the results showing how many times (out of a total 16) the ABF of each control method is significantly better than those of static and random and how many times its one best run is better than the best run of static and random.*

	Rand	PRAM	ME	RL-D	RL-IP	RL-SMC
ABF >static	9	7	6	8	8	7
Best >static	12	10	9	12	12	12
ABF >random		2	3	5	4	0
Best >random		6	6	10	11	9

Table 6.8: *RL-IP / RL-SMC vs RL-D: ++ (or --) shows that the ABF is significantly better (or worse) while + (or -) shows that ABF is not significantly different but the result of the best run is better (or worse). A = means no difference. Grey cells denote that the EA is particularly fit to the specific problem (see explanation in the text).*

Simple ES	+ / --	+ / -	+ / ++	-- / --
Cellular GA	- / --	-- / --	-- / --	-- / +
GA-MPC	+ / -	+ / --	- / +	+ / +
CMA ES	= / =	= / =	-- / +	+ / -

As we have seen before (Section 6.1) random variation can be a surprisingly effective control method, being among the most effective controllers of this experiment (Table 6.7). In more than half of the cases it had significantly better performance than static default parameter values while only in one case it was significantly worse. PRAM also had good results with several EA and problem combinations and outperformed all other controllers for two problems of the Cellular GA (though it performed bad with the other two of the same EA). Overall, PRAM was slightly worse than the RL methods and random variation in terms of success ratio (Table 6.7). An entirely different control method was also tested in this experiment, i.e. meta-evolution. Though our approach on meta-evolution was simple (out-of-the-box CMA-ES), its results were good, especially with two problems of the SES where ME outperformed the other control methods (see Table 6.6). However, it is less consistent since it was unable to make any improvements with the Cellular GA (which was

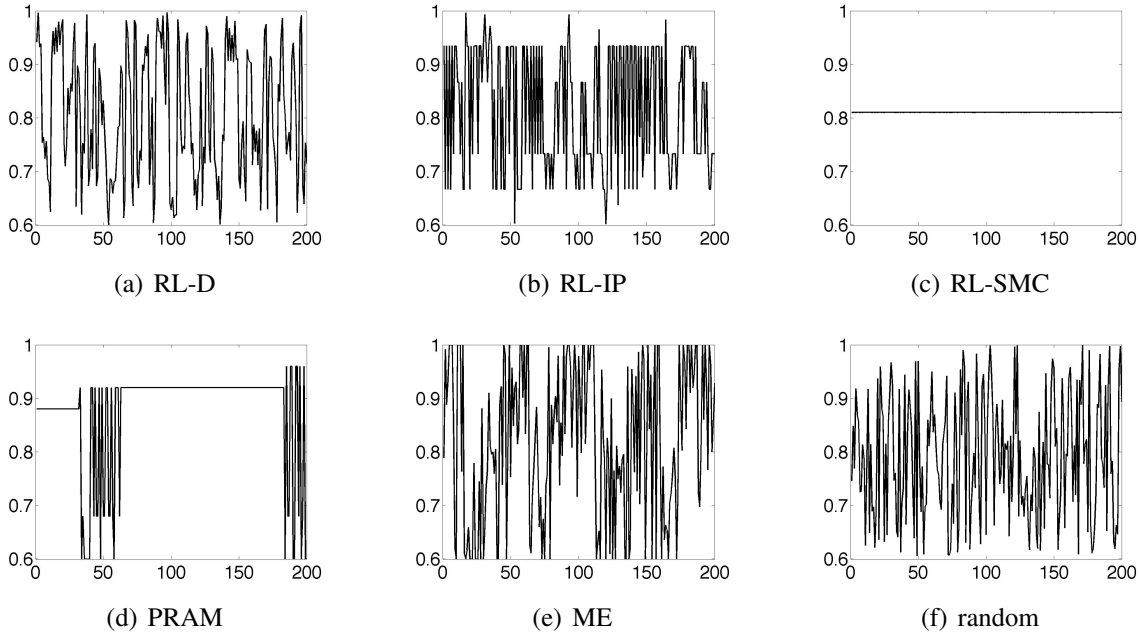
the second easiest for the other methods) and was overall the least effective control method (Table 6.7).

The overall comparison of all the control methods tested here also shows another detail. The "blind"[9] methods (i.e. PRAM and meta-evolution) preformed worse than the RL methods which use the information of the state of the solver. This may be an indication of the importance of feedback for the control process though we cannot generally conclude that from these results.

Looking at the parameter values over time for all controllers it is hard to discern meaningful patterns or explain performance differences with simple rules, though certain characteristics are visible. As an example, Figure 6.3 shows a parameter over time for a specific EA and problem. These figures are representative for each controller; random is included as a comparison. The best performing RL-D and RL-IP are very noisy, partially due to their exploration strategy. RL-D seems closer to random, with the difference of short relatively stable periods, while RL-IP takes less values (possibly the Q values have converged or the dynamic state segmentation behaves differently). On the other hand, RL-SMC has converged to a single value which brings forward one of the disadvantages of a sample based approach. ME is also very noisy, though there appear to be some more obvious trends when compared to RL-D. FInally, PRAM's behaviour is divided into exploration and exploitation periods as expected. The exploitation periods only use one of the values, which points to a weakness in the controller's concept. Improvements in evolution are rare events (isolated jumps) especially near the end, thus, when testing all three current values during an exploration mode, only one of them might get a non-zero value and that will be the only value used in the subsequent exploitation phase.

Finally, though noise seems to be an important factor for a successful control policy, looking at the performance (Table 6.7) and parameter behaviour (Figure 6.3) of RL-D, ME and random variation shows that of course it is not sufficient and it is not yet clear how it affects performance (e.g. compare RL-D to random and random to ME).

[9] Not using observables.



(a) RL-D (b) RL-IP (c) RL-SMC

(d) PRAM (e) ME (f) random

Figure 6.3: *The values of the p_c parameter of the Cellular GA solving f_{23} during the last 200 iterations. The best run of each control method was taken.*

6.2.4 Discussion

The experiment in this section has shown that a direct handling of the parameter space as a continuous action space by the controller does not necessarily provide an advantage in terms of performance. We have tried two alternative continuous designs that gave inferior results compared to the initial discrete RL-D controller. Nevertheless, though we have used two very different approaches for the design of the continuous controllers, they do not cover the whole range of potential methods. Furthermore, the retained dynamic state space segmentation might be imposing limitations or cooperating poorly with the new action space handling parts. Later in this chapter we will examine a fully continuous method that directly handles both the state and action spaces as continuous using function approximation.

In this experiment we have tested and compared a number of control methods relatively large for the typical publications in the field. Further such work can perhaps contribute in parallel to contests and comparison publications for EAs, i.e. it could help users identify which controllers are the good choices for certain frequently appearing problem types as well as stimulate research, and the design of new and better performing control methods.

6.3 Examining the Reward Definition

So far, this chapter has introduced and evaluated several generic parameter controllers based on Reinforcement Learning. All the designs presented have used the same reward definition, i.e. the improvement in best fitness since the last generation (see Eq 6.1 in Section 6.1). This was chosen as the most intuitive option, however, it is not the only possible reward that could be employed for an off-the-shelf generic parameter controller. In this section we focus on the reward definition and examine multiple alternatives to determine what the influence of the reward is on the controller and if there is a specific reward definition that provides superior results compared to the others.

In the existing literature using RL as a parameter control mechanism several alternative rewards have been applied. Muller et al. [212] used temporal difference learning to control the mutation step size for real valued Evolution Strategies (a (1+1)-ES) using success rate (from the 1/5 rule [245]) to define state. They tested four reward types: (i) 1, 0 or -1 if the success rate increased, remained the same or decreased respectively, (ii) the difference of the current fitness minus the fitness of the previous step, (iii) 1, 0 or -1 if the fitness improved, remained the same or deteriorated respectively and (iv) the realised step length in the search space multiplied by the reward as defined in (iii). They reported better results with the latter (iv) and hypothesised that it is because it best approximated the theoretical progress rate for a (1+1)-ES. Pettinger and Everson [223] suggested the RL-GA, a Genetic Algorithm (GA) with an RL agent to control the selection of mutation and crossover operators. Actions are taken per offspring creation, thus rewards are calculated per operator application. They defined reward as the difference of the fitness of the best parent minus the fitness of the best offspring. This difference is normalised by the fitness of the best parent. They report that experiments with alternative reward functions showed that this reward type was the most suitable and stress that penalisation of actions that decrease fitness is important. However, neither the alternative reward definitions nor the relevant results are shown in the paper. Chen et al. [50] presented a modified RL-GA, the SCGA that replaces Q-Learning with SARSA but retains the same reward definition. A modification of this same reward definition was also used by Sakurai et al. [237]. They suggested dividing the reward with the computation time needed by the operator and, also, provided a definition for an aggregated reward on population level that is either the average or the maximum of the rewards received for individual applications of an operator. Considering variation operator control, a reduced version of RL (the multi-armed bandit framework) has been extensively

used for adaptive operator selection (see Section 3.2.2 for a list). Since the selection of a variation operator takes place on the basis of each new offspring produced, reward is often calculated as the improvement in the fitness of the child(ren) as compared to the parent(s) or a reference fitness of the whole population, for example [94] and [112]. Finally, in the area of generic (parameter independent) control, Eiben et al. [73] proposed a generic RL controller using temporal difference learning; they defined reward as the improvement of the best fitness values (following [212]).

Most of the approaches described above (except the last one) were defined for parameter specific controllers and use parameter specific calculations, thus are not directly applicable to our test case in this paper. However, generalised versions, when possible[10], are considered here. These are a fixed reward and a measure of fitness improvement (using a reference or not). This section discusses a number of alternative rewards and their characteristics (simplicity and effectivity). The experiments that follow address the question whether there is a generally good reward function that outperforms others and makes a given EA better than its 'standard' version (with default parameter values). We consider all three different RL mechanisms introduced so far and four reward functions that lead to twelve combinations for the parameter control mechanism. These are compared using four different EAs, each applied to several problems. We run all RL and reward function combinations with each EA and problem combination and compare them in terms of performance, sensitivity, and simplicity.

6.3.1 Reward Definitions

According to Sutton and Barto [263], the reward function for a RL agent should indicate the final goal to be achieved without any biased knowledge about "how" injected by the designer. In the case of controlling the parameters of an EA, the final goal is of course to yield a final fitness as good as possible. Other measures (such as average fitness or diversity) might also be considered relevant, however, they do not necessarily reflect the actual goal and they can introduce a bias in the controller's strategy. Here, we adopt Sutton and Barto's guideline and base our reward definitions on the final best fitness. Of course, that information is not available before the end of the run and, in the case of an one-off problem, the controller has only one single run to learn and apply a good parameter control

[10]Notice that, for example, as discussed above, the reward definition that Muller et al. [212] found most efficient cannot be generalised.

policy[11]. Thus, we require a reward that is calculated after each parameter change (e.g. iteration of the EA) from the beginning and throughout the run but, at the same time, is able to approximate the goal of an optimal solution achieved by the end of the run. We define the following four reward functions (assuming a minimisation problem):

– *Improvement of current best fitness* Δ_f: The most straightforward idea is to use the improvement in fitness since the last action (parameter change). To make rewards more proportionate (EAs make big improvements in the beginning of a run as compared to the end) a ratio is used, though the result is still scale sensitive (the scale of the reward depends on the scale of the improvement). Scale sensitivity can be seen as a potential problem considering how easy huge improvements in fitness are in the beginning of a run and how hard are tiny advances near the end. The improvement ratio is divided by the number of evaluations spent during the last generation (for cases where the population size or number of offspring are among the parameters controlled). The reward is calculated as

$$R(s_t, a_t) = \Delta_f = C \cdot \frac{\frac{f_B^t}{f_B^{t+1}} - 1}{Evals_{t+1} - Evals_t} \tag{6.14}$$

where f_B^t is the best fitness in the population at time t, $Evals_t$ is the total number of evaluations spent up to time t and C is a scaling constant.

– *Binary* 0/1: A very simple reward function that gives 1 when any improvement is made and 0 otherwise.

$$R(s_t, a_t) = \begin{cases} 1, if f_B^{t+1} < f_B^t \\ 0, otherwise \end{cases} \tag{6.15}$$

where f_B^t is the best fitness in the population at time t. This function is completely scale insensitive since the reward is constant and independent from any fitness values. However, because positive rewards are constant, it cannot distinguish between optimal and suboptimal actions.

– *Weighted improvement of current best fitness* $W(\Delta_f)$: This reward is inspired by reinforcement comparison [263], Δ_f from Eq 6.14 is trimmed by a reference reward.

[11] If, on the contrary, we consider a controller for a repetitive application, we could train the controller off-line using multiple training runs, thus being able to directly use the final best fitness of each run as the reward.

$$R(s_t, a_t) = W(\Delta_f) = \Delta_f - Ref(N) \tag{6.16}$$

where Δ_f is defined in Eq 6.14 and Ref is the average of the N last non-zero Δ_f[12]. This reward is a middle solution between the previous two. It is less scale sensitive than the simple Δ_f, thanks to the reference reward, but, unlike the binary, it is also able to discriminate between an optimal and a suboptimal action.

– *Raw current best fitness* f_B: Finally, a very straightforward option is to define reward as the best fitness found in the population.

$$R(s_t, a_t) = -f_B^{t+1} \tag{6.17}$$

This reward has the good property of monotonously increasing (if the EA is elitist and non-restarting), thus giving higher rewards to most recently successful actions. Since it relies on absolute fitness and not relative improvements, the issue of scale sensitivity is not applicable here.

These four reward functions differ in scale sensitivity while some also introduce meta-parameters. We can also assign a level of simplicity to each (as a concept, implementation and applicability). The 0/1 reward is the simplest concept with the widest applicability (it is the only one usable when EA individuals cannot be directly evaluated but only compared). The raw best fitness is also quite simple as a concept and implementation. On the other hand, the Δ_f and $W(\Delta_f)$ rewards are deemed less simple as concepts and implementations. A summary is given in Table 6.9.

Table 6.9: *The defined reward functions along with certain characteristics.*

	Scale Sensitivity	Simplicity	Meta-parameters
Δ_f	High	Less	Scaling factor C
0/1	None	More	None
$W(\Delta_f)$	Moderate	Less	Scaling factor C, window size N
f_B	None	More	None

[12]We did not use the more intuitive ratio $\frac{\Delta_f}{Ref(n)}$ because preliminary experiments showed the difference $\Delta_f - Ref(N)$ to perform better.

6.3.2 Experimental Setup

This experiment aims at comparing the previously defined rewards and determining if a generally better approach for calculating rewards can be identified. Regarding the EAs and problems used, the setup is the same as in the previous Section (see 6.2.2). Here, three RL controllers are applied to each of the EA/problem combinations and for each controller/EA/problem setting all the different reward functions defined in this section are tried.

To evaluate the different reward functions, we apply the three RL-based parameter controllers described previously in this chapter. They are based on Temporal Difference learning and a dynamic segmentation of the state space; they differ mostly in the way they treat the action (i.e. EA parameter) space.

i. The discrete RL-D controller that discretises parameters into fixed intervals (see Section 6.1). The number of discretisation bins was chosen for each EA separately to yield a reasonable number of control actions.

ii. The RL-IP controller that uses interpolation of fixed points in the parameter space (see Section 6.2). The number of points per parameter (resolution) was set for each EA separately for the same reasons as setting the bins for RL-D.

iii. The RL-SMC controller that uses a sample of points (see Section 6.2). For all RL-SMC experiments, the sample size was 40.

For the meta-parameters of these controllers and their values refer to Section 6.2.2 and Table 6.5.

The four reward functions discussed in this section were tested with all controllers and EA/problem combinations. We are interested in how the performance of a controller/reward combination compares to the performance yielded when the EA is run without any parameter control, thus, we also run every EA/problem combination with the EA's parameters static and set to their default values (listed in Appendix B). In this setting, we are interested in achieving maximal performance given a certain amount of effort, thus, the termination criterion of all runs was reaching the maximum number of evaluations ($15 \cdot 10^4$ for all runs with the GA-MPC and 10^6 for all other runs). Each algorithm/problem/controller/reward combination was run 30 times. For the RL-IP controller we used a resolution of 5 when controlling the SES and a resolution of 4 for the other EAs. For the RL-D controller we used 5 action bins when controlling the SES and 4 action bins for the other EAs. For the control ranges of each parameter of all EAs refer to Section 6.1.2 and Table 6.2.

6.3.3 Results and Analysis

Tables 6.10, 6.11 and 6.12 shows all results in terms of final best fitness averaged over all repeats and final fitness of the best run. For every EA/problem combination there are two more variable dimensions: the controller and the reward function used. Subsequently, there is one (independent) table for each of the controllers with subtables for every EA, while each row of every subtable shows the results for a specific reward function. In Section 6.2 we have performed a comparison between the controllers; here we focus on the results from the perspective of reward functions, and attempt a general comparison of the four rewards across different controllers, EAs and problem settings. All significance tests in these results were performed using a two-sample Kolmogorov-Smirnov test (see Appendix B).

For each EA/problem/controller setting in Tables 6.10 through 6.10, the best performing reward function(s) are denoted as well as those that perform significantly better than default static values (the performance of the latter is also shown for comparison). As we have seen previously, the RL controllers are successful (i.e. significantly better than using static default parameter values) when the algorithm is not very refined or particularly tailored to the problem (Sections 6.1 and 6.2). The results here extend this (in general) to the new reward definitions as well. However, they do not show a clear winner among reward definitions or, at least, a pattern of which reward would be better for specific problems or algorithms or controllers (perhaps with the the exception of the $W(\Delta_f)$ being particularly suitable to the SES when controlled by the RL-IP).

A better overview of the results is given in Table 6.13 showing the success count of each reward (i.e. how many times it is significantly better than default static parameter values) and the winning count (i.e. how many times it is the best or not significantly worse than the best reward) for each controller. We are mostly interested in the former count since the target application of the controllers is one-off problems where control aims at improving the performance of a single run that would otherwise use default static parameter values. The winning count can be seen as a secondary measure. We see that the binary reward definition is slightly better for the RL-D and RL-IP controllers, while for RL-SMC Δ_f seems to be better. Though these differences are not very pronounced, the RL-D controller seems to be clearly improved when using the binary reward. Overall, we can say that Δ_f and 0/1 are generally the best performing of the rewards tried here.

Subsequently, we compare the two best performing reward functions (Δ_f and $0/1$) and try to understand why they perform equally well. We would assume that scale sensitivity

Table 6.10: *Results of the experiments with RL-D. There is a section for each EA showing the performance with the four reward types tested in terms of final fitness averaged over all repeats and the final fitness of the best run. Underlined values are significantly better than static default (in grey) and bold values denote the winner(s) (not significantly worse than the best). Significant difference was decided by a two-sided Kolmogorov-Smirnov test with $\alpha = 0.05$. All problems are minimisation.*

Simple ES

	-Rastrigin-		-Schwefel-		-Schaffer-		-FP-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	27.68	20.72	817.53	236.98	22.04	6.93	7935.47	1982.58
Δ_f	**0.67**	0.04	779.27	0.00	9.85	1.08	**166.21**	3.42
$0/1$	**0.45**	0.05	**113.18**	0.00	**4.72**	1.01	**294.85**	13.80
f_B	1.94	0.21	**7.90**	0.00	**1.80**	1.33	168.62	27.36
$W(\Delta_f)$	2.94	1.10	11.20	0.00	2.16	1.50	257.08	93.83

Gellular GA

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	40.89	40.81	-998.83	-999.09	9.06	8.57	**471.81**	435.15
Δ_f	40.80	40.78	-999.18	-999.26	**8.49**	8.15	**484.62**	412.93
$0/1$	**40.78**	40.78	**-999.28**	-999.30	**8.39**	7.96	505.27	414.89
f_B	41.07	40.85	-998.91	-999.21	8.68	7.99	490.40	387.43
$W(\Delta_f)$	40.99	40.85	-998.49	-999.00	8.81	8.37	490.99	409.24

GA-MPC

	-CEC2011 f_7-		-CEC2011 $f_{11.8}$-		-CEC2011 f_{12}-		-CEC2011 f_{13}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	1.68	0.84	1945573	1510399	**16.24**	14.11	**15.25**	8.61
Δ_f	**0.96**	0.63	**947905**	941267	**16.42**	12.59	19.38	14.12
$0/1$	**0.98**	0.67	**948013**	939707	**16.74**	14.16	19.95	12.88
f_B	**0.92**	0.59	**980303**	941593	**16.01**	12.84	18.33	8.76
$W(\Delta_f)$	**0.89**	0.50	**947568**	941161	**16.75**	12.41	18.66	10.36

IPOP-10DDr

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**41.39**	40.78	**-997.67**	-999.31	**6.89**	6.87	**131.54**	108.03
Δ_f	**41.69**	40.78	**-997.75**	-999.31	6.90	6.88	**128.91**	106.93
$0/1$	**41.47**	40.78	**-997.21**	-999.31	**6.91**	6.87	**125.38**	106.89
f_B	41.85	40.78	**-998.30**	-999.31	**6.90**	6.87	**126.93**	105.27
$W(\Delta_f)$	**41.59**	40.78	**-997.27**	-1000.00	6.91	6.87	**125.44**	106.32

would impact performance but Δ_f and $0/1$ lie on opposite extremes in terms of scale sensitivity - see Section 6.3.1. To better understand these two effects, we examined the Q values over time since this is the part of the RL mechanism that is affected by the reward definition. In specific, every state-action pair has a Q value that estimates the long-term return expected if this specific action is taken when in this specific state. Whenever a state-action

Table 6.11: *Results of the experiments with RL-IP. There is a section for each EA showing the performance with the four reward types tested in terms of final fitness averaged over all repeats and the final fitness of the best run. Underlined values are significantly better than static default (in grey) and bold values denote the winner(s) (not significantly worse than the best). Significant difference was decided by a two-sided Kolmogorov-Smirnov test with $\alpha = 0.05$. All problems are minimisation.*

Simple ES

	-Rastrigin-		-Schwefel-		-Schaffer-		-FP-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	27.68	20.72	817.53	236.98	22.04	6.93	7935.47	1982.58
Δ_f	2.27	0.00	708.24	0.00	4.47	0.70	529.67	70.05
$0/1$	4.17	1.18	70.46	0.00	5.04	2.04	444.40	138.69
f_B	4.28	1.66	800.39	236.88	2.26	0.80	79.72	6.60
$W(\Delta_f)$	**0.25**	0.00	**34.88**	0.00	**1.13**	0.41	**46.68**	0.57

Cellular GA

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	40.89	40.81	-998.83	-999.09	9.06	8.57	**471.81**	435.15
Δ_f	40.81	40.78	-999.05	-999.26	**8.60**	7.95	510.29	397.63
$0/1$	**40.78**	40.78	**-999.25**	-999.29	**8.62**	8.12	**476.64**	414.72
f_B	40.83	40.78	-997.94	-998.73	8.74	8.08	**469.77**	410.98
$W(\Delta_f)$	40.87	40.78	-998.78	-999.17	**8.48**	7.83	**481.82**	402.74

GA-MPC

	-CEC2011 f_7-		-CEC2011 $f_{11.8}$-		-CEC2011 f_{12}-		-CEC2011 f_{13}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	1.68	0.84	1945573	1510399	**16.24**	14.11	**15.25**	8.61
Δ_f	**0.93**	0.62	**997692**	940666	17.66	13.68	18.73	8.69
$0/1$	0.99	0.54	**982950**	940537	**17.27**	14.20	18.56	10.98
f_B	**0.99**	0.71	**1039461**	939621	**16.93**	9.79	19.10	10.46
$W(\Delta_f)$	**0.92**	0.74	**949211**	941480	17.18	13.52	19.61	8.86

IPOP-10DDr

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**41.39**	40.78	**-997.67**	-999.31	**6.89**	6.87	**131.54**	108.03
Δ_f	**41.45**	40.78	**-998.17**	-999.31	6.93	6.88	**126.88**	106.30
$0/1$	**41.62**	40.78	**-996.95**	-999.31	6.93	6.87	**127.64**	106.38
f_B	**41.42**	40.78	**-998.38**	-999.31	**6.90**	6.87	**127.44**	106.59
$W(\Delta_f)$	**41.66**	40.78	**-998.42**	-999.31	6.93	6.87	**127.36**	105.92

pair occurs, its Q value is updated using the immediate reward received (the exact formulas for these updates are given in Section 6.1). We would expect a scale sensitive function to create higher Q values and, especially, very high peaks at the beginning of a run as large fitness improvements are typically made in the beginning of the run. Figures 6.4 and 6.5 show examples of the normalised maximum Q value (both over all states and only for the current

Table 6.12: *Results of the experiments with RL-SMC. There is a section for each EA showing the performance with the four reward types tested in terms of final fitness averaged over all repeats and the final fitness of the best run. Underlined values are significantly better than static default (in grey) and bold values denote the winner(s) (not significantly worse than the best). Significant difference was decided by a two-sided Kolmogorov-Smirnov test with $\alpha = 0.05$. All problems are minimisation.*

Simple ES

	-Rastrigin-		-Schwefel-		-Schaffer-		-FP-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	27.68	20.72	817.53	236.98	22.04	6.93	7935.47	1982.58
Δ_f	**8.08**	0.06	488.34	118.44	7.12	1.68	**1027.30**	16.70
$0/1$	**9.38**	0.73	**59.89**	0.00	**3.13**	1.66	**1334.45**	19.96
f_B	**11.10**	0.74	**7.94**	0.00	**3.58**	1.55	**1833.40**	58.14
$W(\Delta_f)$	**9.81**	1.85	**30.96**	0.00	**3.71**	1.79	**1324.91**	34.76

Cellular GA

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**40.89**	40.81	**-998.83**	-999.09	9.06	8.57	**471.81**	435.15
Δ_f	41.62	40.78	-998.62	-999.54	**8.83**	7.83	498.95	403.14
$0/1$	41.53	40.78	-998.41	-999.29	**8.89**	7.79	514.02	405.64
f_B	41.45	40.78	-998.08	-999.26	**9.02**	8.36	525.89	397.06
$W(\Delta_f)$	42.25	40.78	-997.28	-999.30	**9.11**	8.10	533.90	416.94

GA-MPC

	-CEC2011 f_7-		-CEC2011 $f_{11.8}$-		-CEC2011 f_{12}-		-CEC2011 f_{13}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	1.68	0.84	1945573	1510399	**16.24**	14.11	**15.25**	8.61
Δ_f	**0.90**	0.63	**1009638**	945698	**16.95**	10.46	**17.83**	11.05
$0/1$	**0.88**	0.52	**1056653**	942417	**16.99**	13.11	**16.78**	8.74
f_B	**0.91**	0.63	**1156982**	940784	**16.75**	12.98	**15.84**	8.62
$W(\Delta_f)$	**0.94**	0.61	1129855	944213	**16.14**	12.19	17.19	8.62

IPOP-10DDr

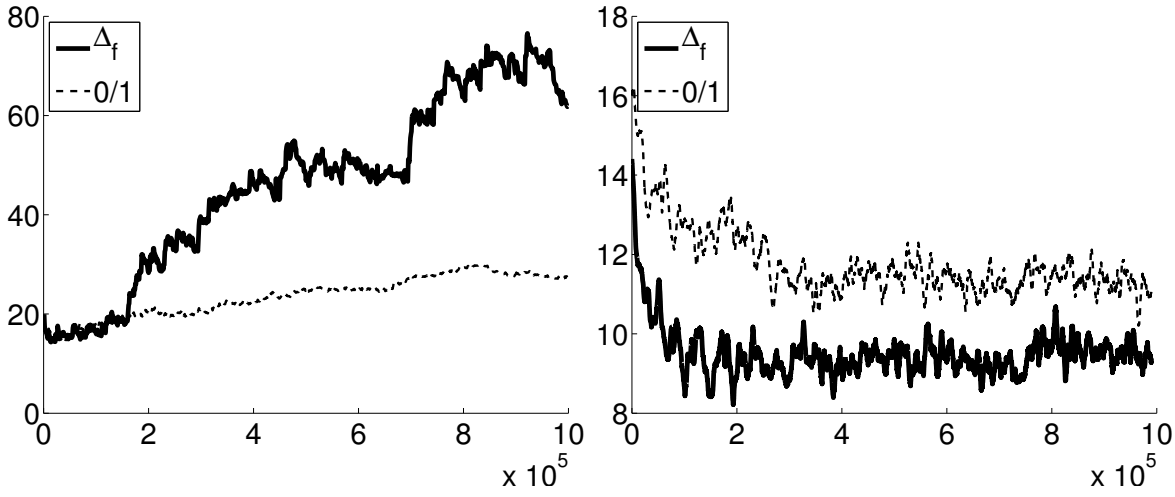
	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**41.39**	40.78	**-997.67**	-999.31	**6.89**	6.87	**131.54**	108.03
Δ_f	**41.45**	40.78	**-998.17**	-999.31	**6.90**	6.87	**126.82**	107.40
$0/1$	**41.52**	40.78	**-997.41**	-999.31	**6.92**	6.87	**129.72**	106.44
f_B	**41.75**	40.78	**-997.71**	-999.31	**6.94**	6.87	**127.68**	106.50
$W(\Delta_f)$	**41.69**	40.78	**-997.00**	-999.31	**6.93**	6.87	**130.83**	108.36

state) over time for two problems of the SES using the RL-D controller. The $0/1$ reward does indeed keep the maximum Q value lower then the Δ_f and that is a consistent observation across settings. However, this impact of the scale (in)sensitivity of the function cannot be correlated to performance: the differences between the different reward types shown in Figures 6.4 and 6.5 are similar but in the former the two reward functions yielded similar

Table 6.13: *Summary of all results from the perspective of reward functions. The table is separated into three sections, one for each controller tested. The two rows denote how many times the result (best final fitness averaged over 30 runs) with each reward is the best (or not significantly worse than the best) and how many times it is significantly better than the performance using default static parameter values.*

	RL-D				**RL-IP**				**RL-SMC**			
	Δ_f	f_i	0/1	$W(\Delta_f)$	Δ_f	f_i	0/1	$W(\Delta_f)$	Δ_f	f_i	0/1	$W(\Delta_f)$
Best	10	8	14	6	6	8	9	11	11	13	13	11
Better than static	8	8	9	7	8	7	9	8	7	6	6	6

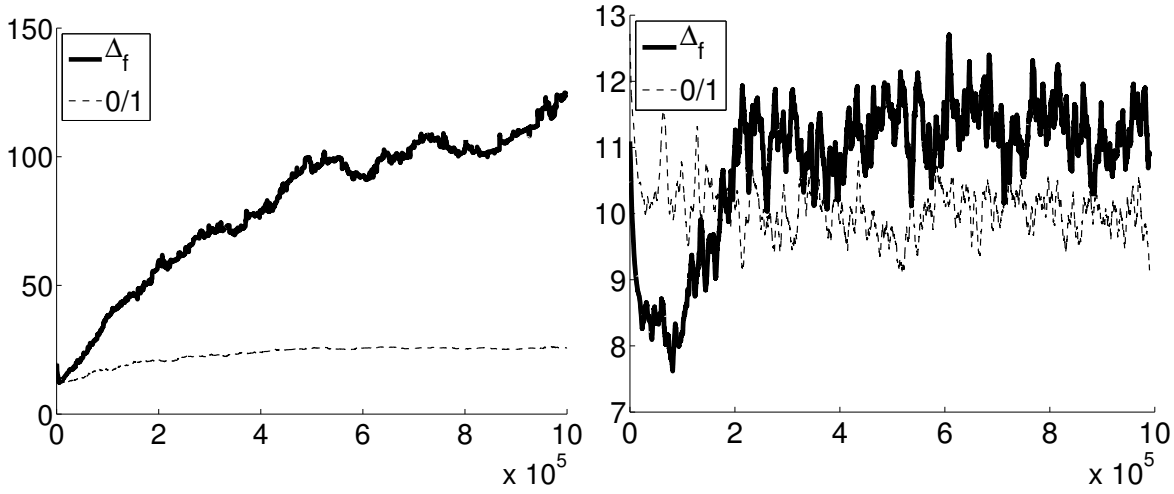
performance while in the latter the binary reward was significantly better. Furthermore, the assumption that scale sensitivity would result in too high rewards early in the run that could have a long lasting misleading effect, does not occur. There is indeed a slight peak of the global maximum Q (Figures 6.4-a and 6.5-a) when using the Δ_f but values quickly decay, while the current maximum Q (Figures 6.4-b and 6.5-b) is not affected.



(a) All (b) Current

Figure 6.4: *The (normalised) maximum Q value taken at each time step as (a) an average of the maximums of all states and (b) the maximum of the current state only. This is the setting of SES, Fletcher & Powell, RL-D. Lines are averages over the 30 runs. The lines for the current state are smoothed with a rolling average of window size 10000.*

From a different perspective, it is interesting why the $0/1$ performs equally well as the Δ_f when it foregoes certain information (the magnitude of improvement). One explanation could be that, in the context of an EA, making any improvement is more important than its magnitude since such an event will give new direction to the search process (escaping a



(a) All (b) Current

Figure 6.5: *The (normalised) maximum Q value taken at each time step as (a) an average of the maximums of all states and (b) the maximum of the current state only. This is the setting of SES, Schwefel, RL-D. Lines are averages over the 30 runs. The lines for the current state are smoothed with a rolling average of window size 10000.*

local optimum). Another influencing factor could be the selection mechanism of the controller, i.e. whether it is greedy or probabilistic; in the latter case we would expect that the magnitude of rewards would influence the selection of actions. However, this does not seem to affect the balance between the Δ_f and $0/1$ rewards: the RL-D uses a greedy selection while the RL-SMC a probabilistic but our results on reward performance were similar for these two. Additionally, we run a control experiment with RL-D using a softmax (i.e. probabilistic) selection instead of a greedy but the results did not show a difference in terms of the comparison between the Δ_f and $0/1$ rewards.

Regarding simplicity and meta-parameters, the $0/1$ reward has an obvious advantage over Δ_f. It is much simpler as an approach (and applicable also to EA cases where individuals cannot be evaluated by a direct fitness value but only comparisons between individuals are possible) and it does not introduce any new meta-parameters. Overall, the $0/1$ reward is the most preferable among the ones tested: it performs equally well or better while being the simplest and without introducing meta-parameters.

6.3.4 Discussion

In this section we introduced various reward functions and evaluated them in different settings. Four reward function were compared: (i) the improvement of the current best fitness; (ii) a binary reward expressing whether an improvement was made; (iii) a weighted improvement function of the current best fitness, and (iv) the raw fitness. Experiments showed that the most simple (i.e. binary) reward function performed better for two controllers and worse for the third. This is surprising as the binary function is unable to distinguish between marginal and good improvements. It can however be considered good news, as a simple parameter-free reward scheme can easily be adopted for generic parameter control, thereby ideally suiting our vision for an out-of-the-box controller that can be applied without any need for tailoring.

6.4 A Fully Continuous Method

As was explained in Section 2.4, when mapping the parameter control problem to reinforcement learning we face a continuous problem, regarding both the state and action spaces. The first controller introduced in this Chapter (the RL-D in Section 6.1) discretises the state and action spaces, the former dynamically on-the-fly and the latter a priori to equal intervals. This simplifies the design since the majority of the existing RL methods deal with discrete problems while continuous problems require special and more complicated treatment [273]. Of course, the a priori discretisation of the action space (parameter ranges) poses an obvious potential disadvantage as was discussed in Section 6.2. To assess that, we replaced the discretisation of the action space with two methods that directly handle the continuous action space.[13] Contrary to our initial expectation, experimental results showed that this slightly impaired performance instead of improving it.

Nevertheless, these results cannot be seen as conclusive regarding the value of using a continuous method for an RL-based parameter controller. First, the continuous action methods used might be inappropriate.[14] Furthermore, the continuous methods used in Section 6.2 were not designed to be used in combination with the dynamic state space segmentation we employed; it may be the case that they simply cannot cooperate well. For a better investigation of the importance of the continuity of the parameter control problem, in this section

[13] While maintaining the same dynamic state space discretisation of RL-D.

[14] For example, in Section 6.2.3 we saw that the limited sample in the RL-SMC can converge to a single value, thus depriving the controller of its ability to constantly learn and adapt.

we employ a fully continuous RL method based on the Continuous Actor Critic Learning Automaton (CACLA) algorithm [274]. CACLA, as its full name implies, is an actor-critic algorithm and uses function approximation[15] to predict the optimal action given a state and to estimate the value of a state (for continuous action and state spaces). Our controller is based on the CACLA design with some modifications, mainly replacing the simple back-propagation neural networks with Extreme Learning Machines (ELMs) [139] (thus we refer to this controller as eXtreme Actor Critic). Details on the XAC design and an experimental assessment follow in this section.

6.4.1 The XAC Controller Design

6.4.1.1 Parameters and Observables

The controller design presented here is parameter-independent and can handle both numeric and symbolic parameters. Updates occur on every iteration (generation) of the EA. The observables used are the same as with the previous controllers (see Section 6.1.1).

6.4.1.2 Algorithm

The eXtreme Actor Critic controller is based on the CACLA algorithm [274] for learning continuous actions in continuous environments. The design involves two neural networks (NNs) (for representing the current policy and for approximating the state value function) and uses the Temporal Difference (TD) error to decide when the policy should be updated (for a short introduction to reinforcement learning refer to Appendix A). The two NNs used are:

– The actor NN represents the current policy π: it receives a state description as input and outputs the action to be taken. Consequently, given N observables and M parameters controlled, the actor NN has N input neurons and M output neurons and makes a mapping $A : \mathbb{R}^N \mapsto \mathbb{R}^M$ corresponding to $a = \pi(s)$. To implement exploration, Gaussian noise is always added to the output of the actor, thus, given observables $\vec{o}(t+1)$, the action taken is

$$a(t) = NN_A(\vec{o}(t)) + N(0, \sigma) \tag{6.18}$$

[15] In specific, neural networks with backpropagation learning.

where σ is the exploration rate and $NN_A(\cdot)$ is the output of the actor NN. In case of symbolic parameters, the output of the corresponding neuron is rounded to the closest integer (the original CACLA algorithm has been used in this manner to solve discrete problems [275])

– The critic NN approximates the state value function V: it receives a state description as input and outputs an estimation of the value of that state. Thus, given N observables, the critic NN has N input neurons and one output neuron making a mapping $C : \mathbb{R}^N \mapsto \mathbb{R}$ that corresponds to $V(s)$.

Of course, NNs need to be trained to perform their predictions and estimations. For the critic NN, every iteration is used as a training example. At every step, the controller receives a set of observables $\vec{o}(t)$ (representing state $s(t)$), takes an action (selected by the actor) and, subsequently, receives the reward $R(t)$ and a new set of observables $\vec{o}(t+1)$ (representing the next state $s(t+1)$). A backup estimation of the value of state $s(t)$ is calculated as

$$\widetilde{V}_{t+1}(s(t)) = R(t) + \gamma \cdot V_{t+1}(s(t+1)) \tag{6.19}$$

where γ is the discount rate and $V_{t+1}(s(t+1))$ is the current estimation of the value of state $s(t+1)$ as is *directly* output by the critic. Every pair $(\vec{o}(t), \widetilde{V}_{t+1}(s(t)))$ is used as a training example for the critic NN. The actor NN is trained using actions that are considered successful. Actions are considered successful when they fulfil two criteria. The first (which was in the original CACLA algorithm) is that the estimated value of a state increases after the action is taken, i.e. when the TD error δ_t is positive:

$$\delta_t = \widetilde{V}_{t+1}(s(t)) - V_{t+1}(s(t)) = R(t) + \gamma \cdot V_{t+1}(s(t+1)) - V_{t+1}(s(t)) > 0 \tag{6.20}$$

where γ is the discount rate, $\widetilde{V}_{t+1}(s(t))$ is the result of the backup operation shown in Eq. 6.19 and $V_{t+1}(s(t))$ is the critic's direct output.[16] The second success criterion is that the last action $a(t)$ led to an improvement in the best fitness of the EA population; this was added to this design since updating the actor with an action that had no effect seemed, in principle, wrong.[17] When these two conditions are met, the pair $(\vec{o}(t), a(t))$ is used as a

[16] Of course, this comparison of Eq. 6.20 is performed *before* the critic is updated with the $(\vec{o}(t), \widetilde{V}_{t+1}(s(t)))$ training example.

[17] In fact, we performed a control experiment without this second success criterion and results showed that it indeed improves the performance of the controller.

training example for the actor, where $\vec{o}(t)$ are the observables of state $s(t)$ and $a(t)$ is the corresponding action taken by the actor (see Eq. 6.18).

To implement the actor and critic we used the Extreme Learning Machine (ELM) approach [140, 139]. This approach involves a typical feedforward neural network with a hidden layer; the hidden weights and biases are set randomly which allows for the output weights to be determined analytically through a linear system when training the network. The ELM approach allows for much faster training than other algorithms, like backpropagation, while obtaining better generalisation (thanks to smaller norms of weights) and smaller errors[18]. The original ELM algorithm is a batch learning method that requires all the training samples at once in order to solve the corresponding linear system; here we use the Online Sequential ELM (OS-ELM) approach [181] that is theoretically shown to have equivalent performance (in terms of error and generalisation) while updating the network chunk-by-chunk or even by one training example at a time. OS-ELM, however, still requires an initialisation phase with a number of distinct training examples at least as high as the number of hidden nodes. Before initialisation the network can only output random values, thus, in the beginning of a run, the XAC actor returns a random action and the critic returns a value of zero[19] until enough training examples are gathered (as explained above) to initialise them. We have used a sigmoid activation function for the hidden layer of both the actor and the critic (the output layers are linear). The output of the actor is enforced within the $[0, 1]$ interval[20] and then mapped to the corresponding parameter value based on the range of that parameter (in case of symbolic parameters the value is rounded to the closest integer as mentioned earlier).

We designed the XAC with a simple meta-control mechanism already in place. As was explained, exploration is performed by adding Gaussian noise to the output of the actor (see Eq. 6.18). The XAC meta-control adapts the exploration rate σ based on the fitness improvement of the EA. In specific, the exploration rate is taken from a range $\sigma \in [\sigma_{min}, \sigma_{max}]$; at every iteration where the best fitness of the EA is not improved σ is multiplied by a factor

[18]It is theoretically shown [140] that, given enough hidden nodes (as many as the training samples), the error will be zero.

[19]If the critic also returned random values that would interfere with the learning process of the actor: as explained above, a training example for the actor is created only when the expectation of the state value is exceeded when taking an action. Thus, the pre-initialisation critic is set to be "pessimistic" to ensure that any good action (with a positive reward) is used to train the actor.

[20]This bounds enforcement is carried out in a "bouncing" manner to avoid getting stuck at the extremes of the interval if exploration becomes aggressive, e.g. an out-of-bounds value of -0.4 will be converted to 0.4 instead of 0 while -1.8 will be converted to 0.2.

f_σ while, when an improvement in best fitness occurs, σ is reset to its minimum value σ_{min}. This mechanism is based on the simple concept that, as the EA is stuck to a local optimum, the exploration of the controller becomes more and more aggressive.

The meta-parameters of the XAC controller are shown in Table 6.14. They include the discount rate for the TD error calculation, the number of hidden nodes for the actor and critic NNs and the chunks used for updating the two NNs. For a discussion regarding the (high) number of meta-parameters and why it is acceptable please refer to Section. 6.1.1.

It is theoretically shown [140] that when the number of hidden neurons is equal to the number of training samples then the NN perfectly approximates the training samples. Of course, here we cannot apply that because it would mean that the NN can only be initialised at the end of a run (remember that the OS-ELM algorithm requires an initialisation with a number of training samples equal to the number of its hidden neurons). For a smaller number of hidden neurons there is no general guideline or theoretic guarantee in terms of error. We chose the number of hidden nodes for the actor and the critic based in the intuition that the more hidden neurons the better performance achieved while keeping them low enough to allow for the NNs to be initialised early enough during the run.[21] The training examples for both NNs are aggregated to small chunks so that the NNs are updates with new samples soon after they are experienced (especially the actor is updated immediately after a training sample is generated). The range of σ is set relative to the range within which the actions are bound (i.e. $[0, 1]$) and is increased slowly (reaching its maximum value in about 550 iteration of stagnation). The discount rate is set moderately high so that learning is relatively far-sighted.

Table 6.14: *XAC controller meta-parameters.*

Parameter	Value	Parameter	Value
Actor hidden N	100	Actor chunk	1
Critic hidden N	200	Critic chunk	10
σ range	$[0.05, 0.15]$	f_σ	1.002
γ	0.8		

[21] Since the critic NN receives a lot more training examples than the actor (only successful actions are used for actor training samples), we set the hidden layer of the critic to be double the one of the actor.

6.4.2 Experimental Setup

The experiments presented in this section aim at evaluating the XAC controller described in the previous section. To this end, we used the same four EAs with their corresponding functions as in the previous experiments of this chapter (see Section 6.2.2). In Section 6.3 we have evaluated a number of different definitions for the reward function of an RL-based parameter controller. In this experiment, we run the XAC and all EA/problem combinations with all four reward functions we have introduced.

Subsequently, we compare the XAC method (keeping the best option for the reward definition) to static parameters set to default values and the results previously acquired with the RL-D controller, which is the most successful RL controller tested so far (the RL-D design is described in Section 6.1 but here we use the results acquired with RL-D and binary reward as shown in Section 6.3). We also compare XAC performance with the usual benchmarks of random variation of parameter values, PRAM and Meta-Evolution (see Appendix B for descriptions of the EAs, problems and controller benchmarks).

As with the previous experiments in this chapter, the termination criterion of all runs was reaching the maximum number of evaluations ($15 \cdot 10^4$ for all runs with the GA-MPC and 10^6 for all other runs). Each problem/controller/reward combination was run with the XAC controller 30 times. The meta-parameters of the XAC are set to the values shown in Table 6.14.

6.4.3 Results and Analysis

The full results of the experiment are shown in Table 6.15. The table is organised from the perspective of a comparison among the different reward definitions; significance test were performed using a two-sample Kolmogorov-Smirnov test (see Appendix B). Contrary to the results we acquired in Section 6.3, here the binary reward does not seem to be the best choice. Table 6.16 shows a summary of the results in terms of success counts (i.e. how many times a reward definition yields results that are among the winners and/or better than static). We can see that the $W(\Delta_f)$ is a better option for the XAC controller in many cases. A reason for this difference could be the fact that the algorithm of the XAC has, by itself, no concept of continuity (in time) or history. The RL-D, RL-IP and RL-SMC controllers use eligibility traces (see Sections 6.1.1 and 6.2.1) to give part of a received reward to previous actions that have possibly contributed to the success of the last action. On the other hand, XAC does not use such eligibility traces so it does not consider previous actions when training

the actor with a successful action. The $W(\Delta_f)$ reward is history based (see Section 6.3.1) and incorporates a comparison of the last action's result to the results of previous actions. The fact that $W(\Delta_f)$ is more successful for XAC, unlike the other controllers, can be an indication that some knowledge of recent past actions can be important when considering the reward and success of an action.[22]

Table 6.17 shows the results of the XAC experiments in comparison to the results of RL-D and the benchmarks. As with the previous controllers, XAC yields a performance improvement when the EA is not tailored to the problem it is solving (though it fails to improve CGA solving f_{21}^{BBOB} which RL-D does). The summary of the comparison in a form of success counts is given in Table 6.18. We can see that the success ratios of XAC are overall slightly worse to that of RL-D. However, with the SES, the XAC controller outperforms RL-D; though their success ratios are equal, when compared directly, XAC has better performance (see Table 6.17). On the other hand, with the CGA, XAC does not perform well: even though it is better than static in two cases it still remains on the level of random variation (while RL-D here is significantly better than random). Finally, with the GA-MPC and IPOP-10DDr, XAC did not manage to outperform the effect of random variation.

A factor influencing the unsuccessful behaviour of XAC with CGA is the number of training examples made available to the actor of XAC during a run. As can be seen in Figure 6.6, the CGA makes the least steps[23] in fitness of all the EAs, which results in the smallest amount of training examples for the actor.[24] In fact, in two of the test functions, the total number of steps is never higher than 100, which is the number of training examples required to initialise the actor; this means that for these problems, XAC was acting entirely randomly. Examples of the relation between the number of training examples generated for the actor the final best fitness achieved are given in Figure 6.7. In some cases (mostly (d) and (f)), a moderate trend for a downward slope can be seen: the more training examples the lower (better) the final fitness; for others (e.g. (a) and (e)), there seems to be no correlation. Cross-checking with the results in Table 6.17 we observe that the presence of such correlation

[22]Notice however, that the eligibility traces and the history based $W(\Delta_f)$ reward approach this matter in different ways: the former gives credit to actions before a successful action based on the idea that an improvement of fitness can be the result of a process longer than just one generation while the latter makes sure that trivial actions (i.e. actions that are successful but, nonetheless, at a point in time when the EA already has a momentum of improvement) receive less reward.

[23]By "step" we mean an improvement in the current best fitness of the population.

[24]As was explained earlier, a training example for the actor is generated after a successful action, i.e. an action that resulted in an improvement of the current best fitness.

Table 6.15: *All results for the XAC controller tested with four reward definitions. There is a section for each EA showing the performance with the four reward types tested in terms of final fitness averaged over all repeats and the final fitness of the best run. Underlined values are significantly better than static default (in grey) and bold values denote the winner(s) (not significantly worse than the best). Significant difference was decided by a two-sided Kolmogorov-Smirnov test with $\alpha = 0.05$. All problems are minimisation.*

Simple ES

	-Rastrigin-		-Schwefel-		-Schaffer-		-FP-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	27.68	20.72	817.53	236.98	22.04	6.93	7935.47	1982.58
Δ_f	**1.01**	0.01	7.93	0.00	2.26	0.65	**79.77**	1.50
$0/1$	**0.77**	0.00	11.85	0.00	2.70	0.83	**39.25**	1.58
f_B	4.08	0.08	19.74	0.00	2.19	0.70	**39.03**	1.40
$W(\Delta_f)$	**0.98**	0.00	**7.90**	0.00	**1.24**	0.45	**79.59**	1.21

Cellular GA

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**40.89**	40.81	**-998.83**	-999.09	9.06	8.57	**471.81**	435.15
Δ_f	**40.87**	40.80	**-998.85**	-999.12	**8.71**	8.22	502.86	446.61
$0/1$	40.97	40.79	-998.68	-999.17	**8.71**	8.22	502.86	446.61
f_B	**40.87**	40.80	-998.69	-999.23	**8.71**	8.22	502.86	446.61
$W(\Delta_f)$	**40.90**	40.79	**-998.83**	-999.19	**8.71**	8.22	502.86	446.61

GA-MPC

	-CEC2011 f_7-		-CEC2011 $f_{11.8}$-		-CEC2011 f_{12}-		-CEC2011 f_{13}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	1.68	0.84	1945573	1510399	**16.24**	14.11	**15.25**	8.61
Δ_f	**0.86**	0.55	**961103**	942691	**16.26**	12.37	17.73	11.33
$0/1$	**0.88**	0.63	**961103**	942691	**16.40**	12.36	**17.38**	8.64
f_B	**0.87**	0.56	**961103**	942691	**16.26**	12.37	17.73	11.33
$W(\Delta_f)$	0.96	0.50	**961103**	942691	**17.12**	12.36	**16.96**	8.64

IPOP-10DDr

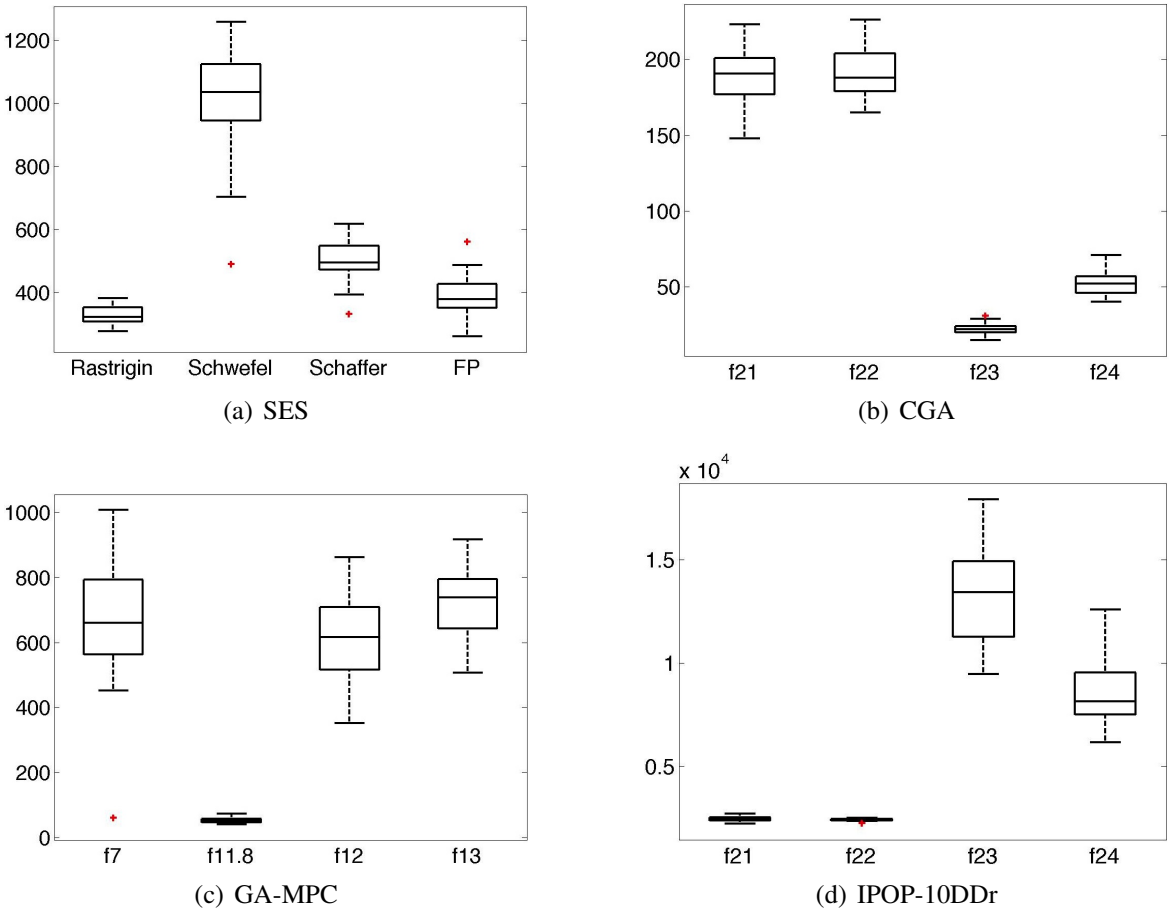
	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**41.39**	40.78	**-997.67**	-999.31	**6.89**	6.87	**131.54**	108.03
Δ_f	**41.59**	40.78	**-998.38**	-999.31	6.93	6.88	**132.84**	107.15
$0/1$	**41.40**	40.78	**-997.96**	-999.31	6.95	6.88	**133.11**	111.22
f_B	**41.13**	40.78	**-998.36**	-1000.00	6.93	6.88	**129.92**	111.00
$W(\Delta_f)$	41.65	40.78	**-997.33**	-999.31	6.93	6.87	**129.79**	106.01

between the number of the actor's training examples and the final fitness does not necessarily imply a successful application of the XAC.

Finally, we examine the values of parameters over time for RL-D and XAC for a setting they both perform well (Figure 6.8). We can see that the two controllers create very differ-

Table 6.16: *Summary of the results comparing reward definitions for the XAC controller. The two rows denote how many times the result (best final fitness averaged over 30 runs) with each reward is the best (or not significantly worse than the best) and how many times it is significantly better than the performance using default static parameter values.*

	Δ_f	f_i	0/1	$W(\Delta_f)$
Best	11	9	10	12
Better than static	7	7	7	8



(a) SES (b) CGA

(c) GA-MPC (d) IPOP-10DDr

Figure 6.6: *The number of fitness improvements per run for all EAs in the experiment when controlled by XAC. For each algorithm, there is one boxplot per test function.*

ent patterns. The XAC controller converges around a value (though noise is still present) resembling an "online tuning" approach; this value is retained even in the face of the SES restarting (the peaks of the fitness curves). RL-D, on the other hand, generally maintains dynamism throughout the run (though we cannot be sure that it is the result of responding to different EA states and not a stochastic effect). The converging behaviour XAC exhibits in Figure 6.8 is not always the case; Figure 6.10 shows a counter-example where XAC remains

Table 6.17: *Results of the experiments evaluating the XAC controller. There is a section for each EA including subtables with the results for each test problem. Subtables show the performance of the EA with that problem in terms of final fitness averaged over all repeats (ABF) and final fitness of the best runs. In the ABF columns, underlined values are significantly better than static, bold values denote the winner(s) (not significantly worse than the best) and asterisks mark values significantly better than random. Significance is decided by Kolmogorov-Smirnov tests with $\alpha = 0.05$. All problems are minimisation.*

SimpleES

	-Rastrigin-		-Schwefel-		-Schaffer-		-FP-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	27.68	20.72	817.53	236.98	22.04	6.93	7935.47	1982.58
random	4.65	1.63	552.13	118.44	3.07	1.98	420.26	187.94
PRAM	11.76	2.18	422.49	0.00	4.25	0.91	1022.72	15.56
ME	**0.12***	0.00	587.15	118.44	1.45*	0.38	79.87*	10.62
RL-D	0.45*	0.05	**113.18***	0.00	4.72	1.01	294.85*	13.80
XAC	0.98*	0.00	**7.90***	0.00	**1.24***	0.45	**79.59***	1.21

GridFPGA

	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	40.89	40.81	-998.83	-999.09	9.06	8.57	**471.81***	435.15
random	40.84	40.80	-998.92	-999.12	8.64	8.30	490.17	422.01
PRAM	**40.78***	40.78	**-999.30***	-999.49	9.28	8.44	493.93	412.91
ME	41.00	40.87	-998.44	-998.89	8.93	8.41	510.60	465.43
RL-D	40.78*	40.78	-999.28*	-999.30	**8.39***	7.96	505.27	414.89
XAC	40.90	40.79	-998.83	-999.19	8.71	8.22	502.86	446.61

GAMPC

	-CEC2011 f_{7}-		-CEC2011 $f_{11.8}$-		-CEC2011 f_{12}-		-CEC2011 f_{13}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	1.68	0.84	1945573.38	1510399.76	**16.24**	14.11	**15.25**	8.61
random	**0.90**	0.58	**946705.40**	939736.56	**16.54**	13.72	**17.63**	8.79
PRAM	1.55	0.59	1580994.58	941087.44	**16.20**	14.23	17.43	8.86
ME	**0.99**	0.77	**951570.52**	939398.63	17.86	13.54	19.63	14.07
RL-D	**0.98**	0.67	**948013.90**	939707.66	**16.74**	14.16	19.95	12.88
XAC	**0.96**	0.50	961103.67	942691.75	**17.12**	12.36	**16.96**	8.64

IP10DDr

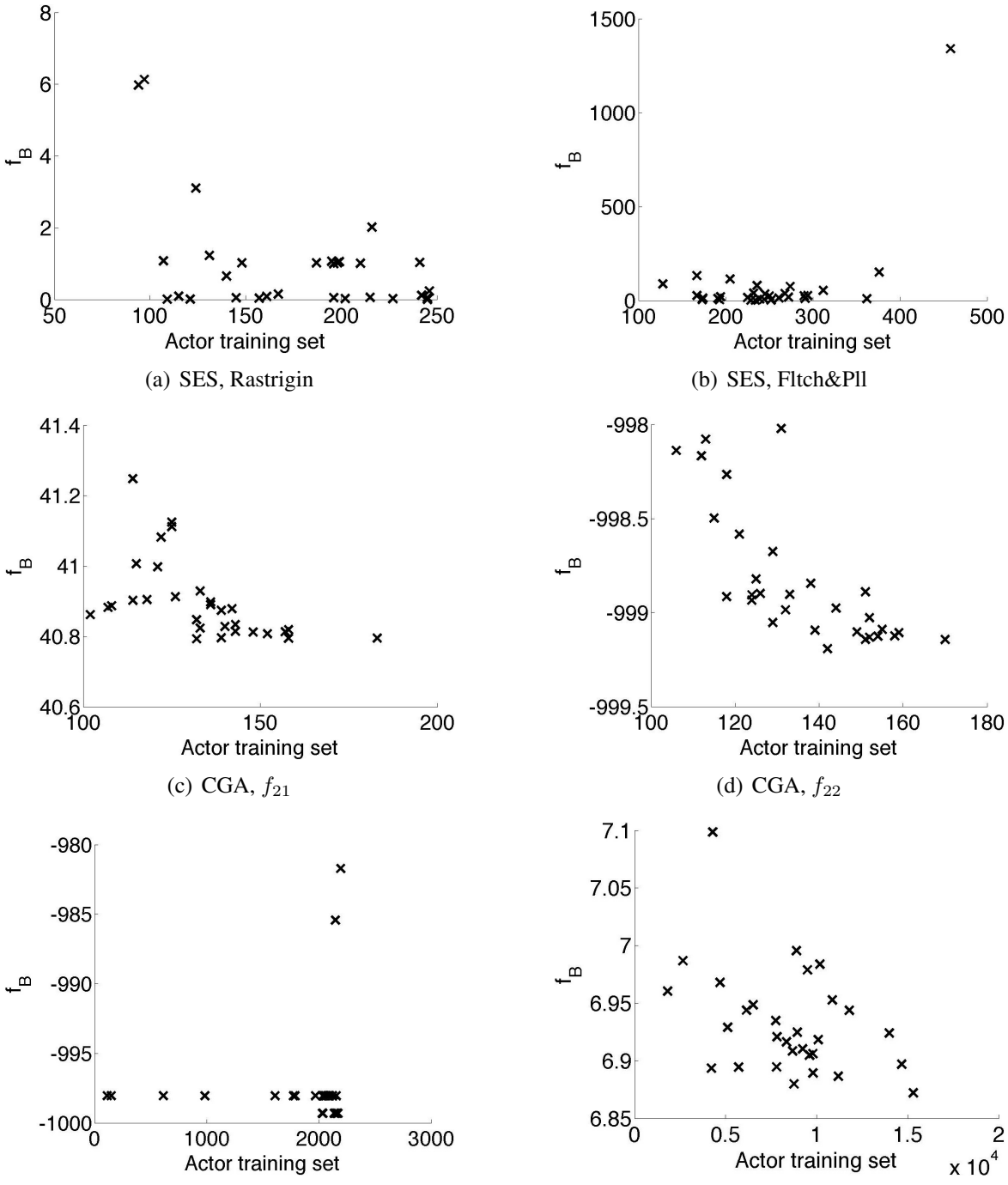
	-BBOB f_{21}-		-BBOB f_{22}-		-BBOB f_{23}-		-BBOB f_{24}-	
	ABF	*Best*	*ABF*	*Best*	*ABF*	*Best*	*ABF*	*Best*
static	**41.39**	40.78	**-997.67**	-999.31	**6.89**	6.87	**131.54**	108.03
random	**41.75**	40.78	**-997.88**	-999.31	**6.94**	6.87	**126.46**	107.75
PRAM	41.99	40.78	**-997.63**	-999.31	6.91	6.87	**137.57**	109.71
ME	**41.17**	40.78	**-997.03**	-999.31	6.92	6.87	**127.81**	106.42
RL-D	**41.47**	40.78	**-997.21**	-999.31	**6.91**	6.87	**125.38**	106.89
XAC	**41.65**	40.78	**-997.33**	-999.31	6.93*	6.87	**129.79**	106.01

Table 6.18: *Summary of the results showing how many times (out of a total 16) the ABF of each control method is significantly better than those of static and random and how many times its one best run is better than the best run of static and random.*

	Rand	PRAM	ME	RL-D	XAC
ABF >static	9	7	6	9	8
Best >static	12	10	9	11	11
ABF >random	-	2	3	6	5
Best >random	-	6	6	10	11

dynamic throughout the whole run. Here, we can clearly see the parameters reacting to the restarts of the EA (especially visible for k_s). This shows that XAC is indeed capable of learning a control policy that takes into account and reacts to the state of the EA.

Perhaps an explanation for the different behaviours of the controllers is their action selection methods. RL-D uses discrete actions with Q values initialised to 1, thus it has a bias towards trying all actions until they are proven bad. On the other hand, the XAC actor starts random and is updated by training examples as they are generated making it focus in specific areas and learn certain mappings. Since the output of the actor itself influences the training instances generated in the future, we can understand that XAC is more prone to prematurely converging to a stable output due to a successful action encountered early in the run and the inability to learn anything further. This is demonstrated in Figure 6.9 showing parameter values controlled by XAC for an EA/problem setting it performed bad. Parameters still converge while XAC's meta-control is unable to increase the actor's exploration rate sufficiently. This is due to the fact that, during the run, several (small) fitness improvements occur (see Figure 6.6-(c)), thus, the exploration rate is constantly reset to its minimum default value and XAC is unable to find better parameter values. Here, it is worth noticing that the size of the training chunk for the actor may be an influential meta-parameter, though that was not obvious beforehand. A larger size may slow learning down but may lead to better generalisation of the actor while a smaller size speeds up the training process but may, as explained, lead to a premature convergence of the controller. In other words, the actor's training chunk size may, in fact, affect the exploration-exploitation balance of the controller.



(a) SES, Rastrigin

(b) SES, Fltch&Pll

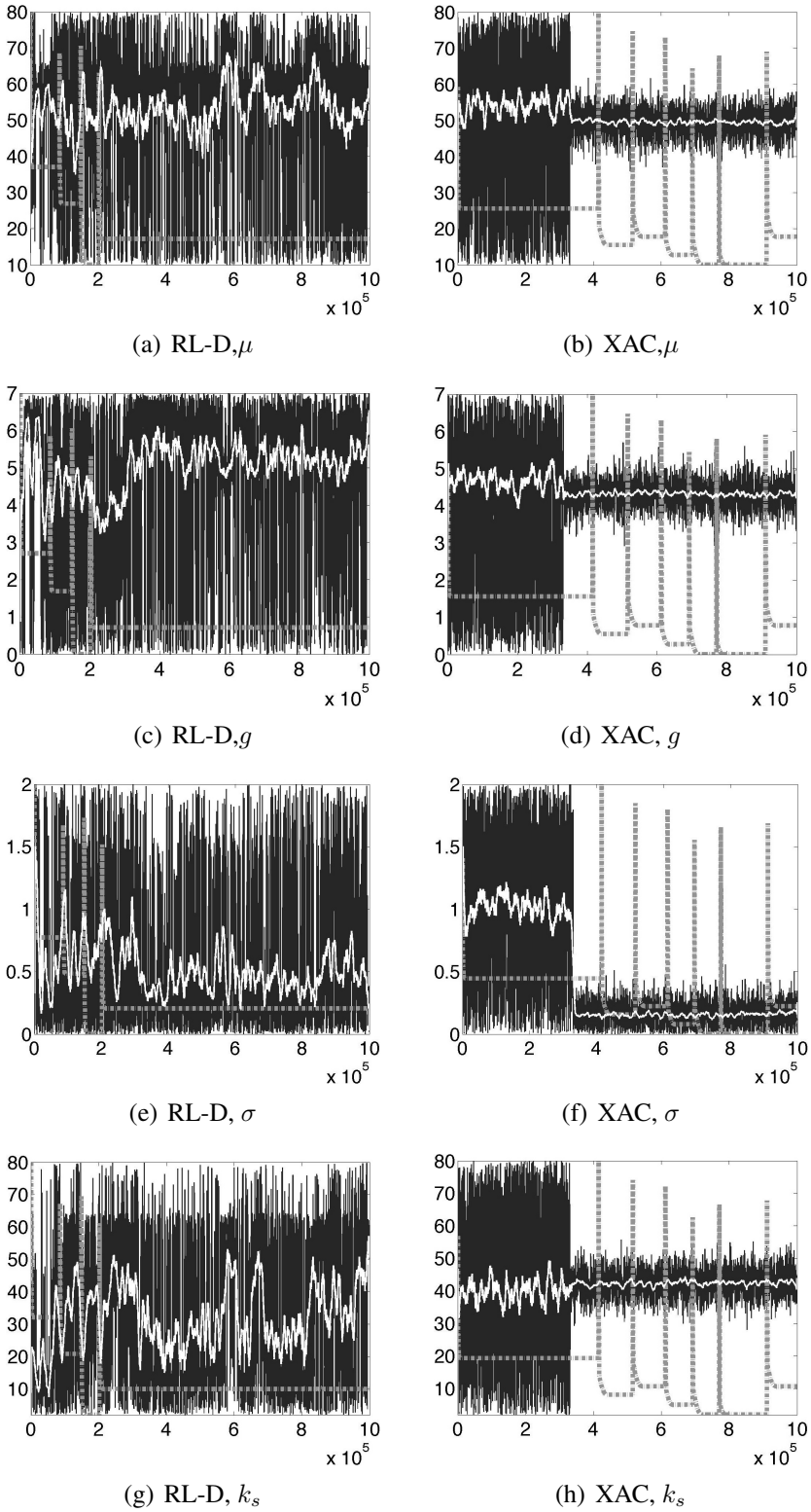
(c) CGA, f_{21}

(d) CGA, f_{22}

(e) IPOP-10DDr, f_{22}

(f) IPOP-10DDr, f_{23}

Figure 6.7: *Six examples of the final best fitness of runs versus the number of training examples produced for the XAC actor during that run. Each plot shows the data for an EA/problem combination with each point representing a run.*



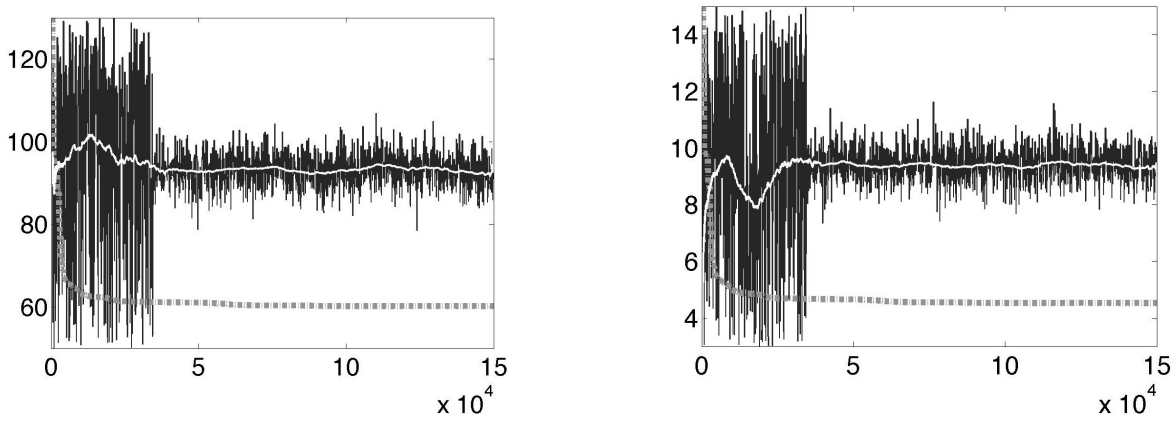
(a) RL-D,μ (b) XAC,μ

(c) RL-D,g (d) XAC, g

(e) RL-D, σ (f) XAC, σ

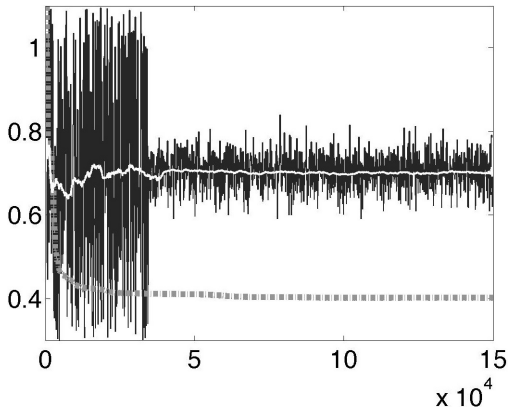
(g) RL-D, k_s (h) XAC, k_s

Figure 6.8: *Parameter values over time controlled by XAC and RL-D for the Simple ES solving the Schwefel function. Values shown are from the best runs of each controller for the specific EA/problem setting. The inner white lines are the smoothed curves with a span of 10000. The grey dotted line shows the current fitness (scaled to the range of the parameter).*

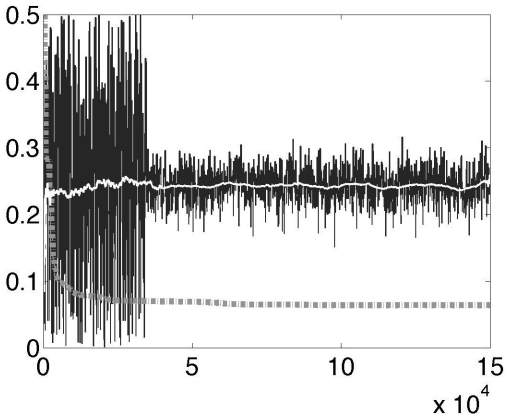


(a) XAC, μ

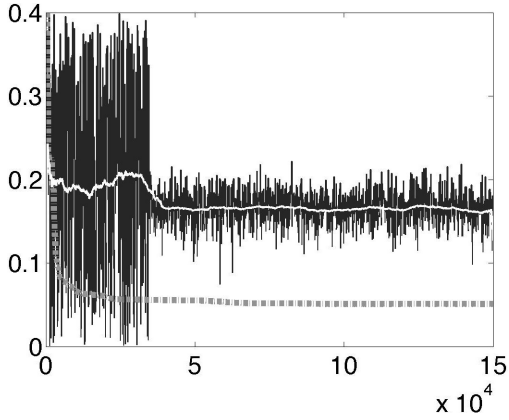
(b) XAC, k_m



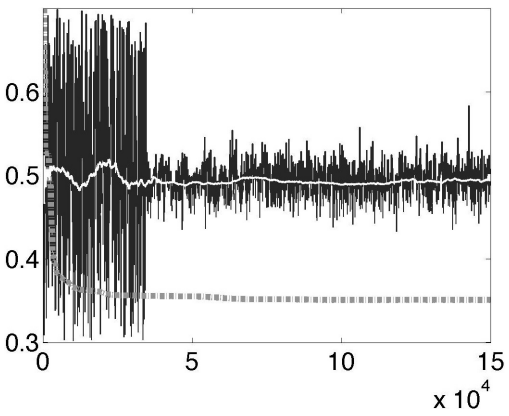
(c) XAC, β_m



(d) XAC, β_{std}

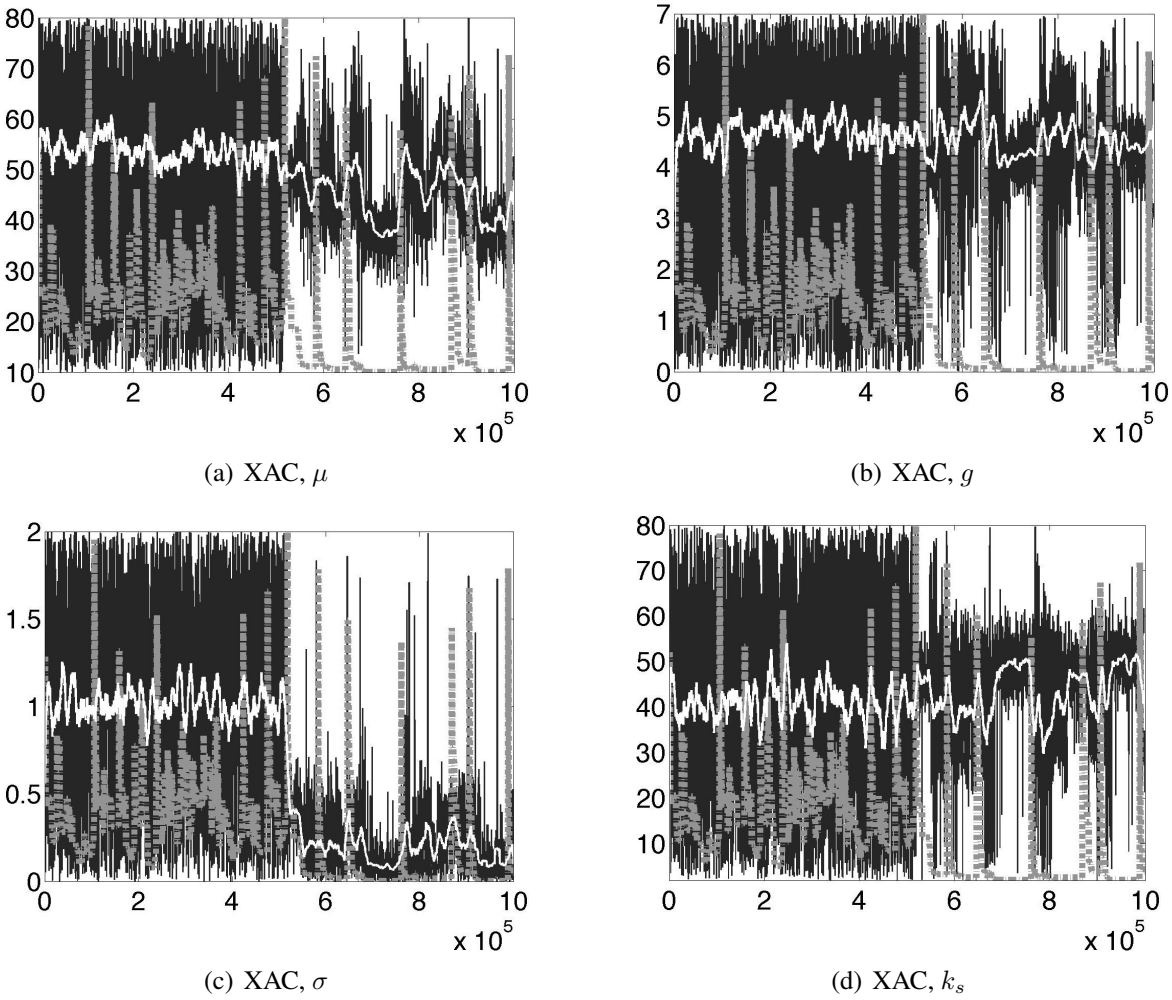


(e) XAC, p_r



(f) XAC, f_α

Figure 6.9: *Parameter values over time controlled by XAC for the GA-MPC solving the f_{13} function. Values shown are from the best run of the specific EA/problem setting. The inner white lines are the smoothed curves with a span of 10000. The grey dotted line shows the current fitness (scaled to the range of the parameter).*



(a) XAC, μ

(b) XAC, g

(c) XAC, σ

(d) XAC, k_s

Figure 6.10: *Parameter values over time controlled by XAC for the SES solving the Rastrigin function. Values shown are from the best run of the specific EA/problem setting. The inner white lines are the smoothed curves with a span of 10000. The grey dotted line shows the current fitness (scaled to the range of the parameter).*

6.4.4 Investigating On-line Tuning

As discussed above, XAC sometimes seems to behave like an on-line tuner, i.e. it converges around a specific value for each parameter and retains this value (with exploration induced noise) throughout the run. This observation points back to the argument supporting that parameter control is beneficial because different parameter values are better at different times during evolution. The results of Chapter 5 supported this argument: in some cases variation alone was able to outperform the optimal (tuned) but static values. On the other hand, the observation that the XAC controller converges to stable values challenges that same argument. If the controller's strategy is maintaining a stable value, perhaps dynamic control is not as beneficial.

To clear out this contradiction we run an additional experiment: we run SES with static parameters set to the values that XAC converged to. If the effect of XAC is merely to find good static values on-the-fly, then this static SES will be at least as good as its XAC controlled counterpart. To derive the "XAC-tuned" parameter values we took the average of the last quarter of the smoothed parameter curve over the evaluations count (examples of such smoothed curves are the white lines in Figures 6.8 to 6.10). For each test function, we used the best run that XAC had with that function. The results are shown in Table 6.19. As was expected, the performance of the static SES greatly improved by using the XAC-tuned values as opposed to the previous defaults. However, in three out of four problems, the XAC controlled EA is still significantly better. This result supports the view that dynamic control can be beneficial but also suggests that it is not necessarily so for every case but can depend on the problem at hand.[25].

Table 6.19: *A comparison of the performance when using XAC control and when running with static parameters set to the values XAC (averagely) converged to in the best run with the specific problem. Bold values denote winners (by Kolmogorov-Smirnov tests with $\alpha = 0.05$).*

	Rastrigin	Schwefel	Schaffer	FP
Static (xt)	6.3587	**0.0018**	4.3544	93.4783
XAC	**0.9806**	7.8976	**1.2371**	**79.5921**

[25]This is an indication and by no means a proof or a generalizable conclusion

6.4.5 Discussion

In this section we have presented the design of a fully continuous controller based on an actor-critic method and have experimentally evaluated its performance. Despite our motivation for a continuous controller (see Section 6.2), XAC did not outperform the discrete RL-D in terms of the success ratio (number of cases it is better than static). However, in the cases that XAC did have a successful impact the results were very good and it achieved better performance than RL-D. This unreliable behaviour may be a result of design flaws or, perhaps, an indication of a trade-off between performance potential and ease of training (bringing to mind the concept of No Free Lunch). Examples of parameter values over time, when controlled by XAC, demonstrate that the controller is capable of learning meaningful control policies that react to the state of the EA.

Though XAC was designed as a generic and EA-independent controller, it actually does impose one requirement: it needs the EA to make a sufficient number of fitness improvements during a run to allow XAC to generate the needed training examples for its actor. An EA that only makes a few jumps in fitness during a run cannot be successfully controlled by XAC. A way to partially alleviate this problem would be to store the successful transitions occurring before the critic is initialised and, subsequently, convert them to training examples for the actor by evaluating the starting and ending EA states of each transition with the, now functional, critic. Furthermore, our analysis showed that XAC can be prone to premature convergence due to the way its actor is trained. The meta-control of the actor's exploration rate was not always able to deal with this problem because even the smallest improvement in fitness causes the meta-controller to reset the exploration rate. Consequently, a more intelligent approach for the meta-controller may improve the results of XAC in certain cases. Also, the size of the actor's training chunk may be an important meta-parameter influencing the exploration-exploitation balance of the controller and may deserve a closer look.. Overall, we can say that there is room for improvement in the design of the controller that may allow it to better utilise its potential as a continuous method.

As a last remark, the observation of cases where XAC exhibited an "online tuner" behaviour and a subsequent control experiment, showed that, though the argument that dynamic control of parameters is beneficial is still supported, it is not necessarily true for all problems.

6.5 Evaluation with Dynamic Environments

A major motivation for parameter control is helping EAs to better deal with dynamic problems. If we accept that different parameter values are better at different times of an evolutionary run then, it certainly makes sense that different values are also better in different situations of an environment. EAs have been applied extensively to dynamic problems [215] and specific techniques have been suggested to deal with such problems, e.g. hypermutation [53][26], random immigrants [115], multiple populations [6] and redundant coding [291].

In this section we evaluate parameter control with dynamic problems (or better with EAs solving dynamic problems). We are interested to see if the changing of the environment itself constitutes an attribute that makes parameter control advantageous. The setting is different from the one-off optimisation application we have been focusing on so far in this chapter: dynamic problems are found in online applications where we are not interested in finding a good solution within a certain amount of time but, instead, in constantly maintaining good solutions and tracking the changing environment throughout a run.

6.5.1 Experimental Setup

Performance measure

As explained above, in dynamic problem settings we are not interested in the best solution found during a whole run be in having fit individuals in the population at all times. Consequently, the measure of average best fitness we have used so far is not relevant here. There are several performance metrics for dynamic problems found in literature [292]. Here we use the online error, i.e. the distance between the fitness of the best individual in the current population from the current target fitness of the landscape.[27] This defines a curve over time; we take the area under this curve as the performance metric of a run:

$$E_O = \int_1^T e(t)dt = \int_1^T |f_B(t) - F_T(t)|dt \tag{6.21}$$

where $f_B(t)$ is the best fitness in the population at time t and $F_T(t)$ is the target fitness of the problem at time t. We use the evaluations' count for defining time instead of generations because the population size of the EAs is among the controlled parameters. To derive an

[26] In fact this is a simple adaptive control mechanism that increases the mutation rate whenever a change in the environment is detected.

[27] The fitness landscape changes with time and so does the value of the optimal fitness possible.

overall score for a controller/EA/problem setting we simple average over all runs for an average online error (AOE) value.

EAs and Problems

For this experiment we use synthetic fitness landscapes because they have the advantage of being customisable. This allows us to test problem instances that differ in terms of frequency and intensity of change while maintaining all other factors (the type and general of the problem) the same. In specific, we experiment with two EAs, a standard implementation of a GA, and a GA with a mechanism designed to cope with dynamic environments (for details of the EAs and the problems refer to Appendix B):

- The Standard Genetic Algorithm (SGA) solving the Dynamic Bit Matching (DBM) problem with a length of 100 bits. We use several instances of the problem ranging in frequency and intensity of change, starting with f_1^{DBM} that has small changes occurring very frequently making it a gradually moving problem to f_4^{DBM} that involves only 10 very drastic changes. For a comparison, we also include f_0^{DBM} that is a static instance of the DBM with no change at all.

- The Random Immigrant Genetic Algorithm (RIGA) solving the Moving Peaks Problem (MPP) in 10 dimensions . As with the previous problem, we tested four instances ranging in frequency and intensity of change, starting with the slow moving f_1^{MPP} to $f_4 MPP$ that has only a few very drastic changes. Again, we include a static instance f_0^{MPP} of the problem.

Detailed settings of the problems and the control ranges of the EA parameters are given in Table 6.20.

Control methods

For this experiment we use RL-D and XAC since they are the most successful of the designs presented in the chapter. We use the same rewards as before for these controllers even though the performance measure we use in this experiment is different; the target of the controller is still to improve the population's fitness as much as possible. RL-D was run with the $0/1$ reward and 4 bins for action discretisation; XAC was run with the history based $W(\Delta_f)$ reward. The meta-parameters for both controllers were set as in the previous

Table 6.20: *Experimental Setup*

EA	Parameters, ranges	Problems
SGA	$\mu \in [10, 100]$ $p_c \in [0, 1]$ $p_m \in [0, 0.2]$ $N \in [1, 99]$ $xover \in \{uni, np\}$ $s \in [1, 2]$	Dynamin Bit Matching, $l = 100$ f_1^{DBM}: $g = 200$, $d = 2$ f_2^{DBM}: $g = 10^3$, $d = 10$ f_3^{DBM}: $g = 10^4$, $d = 30$ f_4^{DBM}: $g = 10^5$, $d = 80$
RIGA	$\mu \in [40, 400]$ $p_c \in [0, 1]$ $p_m \in [0, 1]$ $N \in [1, 9]$ $xover \in \{uni, np\}$ $k_s \in [1, 400]$ $r_i \in [0, 0.2]$	Moving Peaks Problem,10 peaks, 10 dimensions f_1^{MVP}: $U = 200$, $s = 0.08$, $C_H = 0.5$, $C_W = 0.05$ f_2^{MVP}: $U = 2000$, $s = 0.5$, $C_H = 3$, $C_W = 0.5$ f_3^{MVP}: $U = 10^4$, $s = 1$, $C_H = 7$, $C_W = 1$ f_1^{MVP}: $U = 5\dot{1}0^4$, $s = 100$, $C_H = 10$, $C_W = 2$

experiments (see Tables 6.1 and 6.14).[28] Each EA/problem setting was also run with static EA parameters set to default values (listed in Appendix B) and with the random benchmark.

6.5.2 Results and Analysis

The results of all the experiment are given in Table 6.21; significance tests were performed using the two-sample Kolmogorov-Smirnov test (see Appendix B). We can see that the controllers achieve significantly better performance in almost all dynamic test cases. The XAC controller has clearly better results than RL-D here, both in terms of success ratios and in direct comparison between the two RL controllers. The fact that the RL controllers are consistently better with almost all the dynamic problem instances, while for the static problem instances (the f_0's) static parameter values perform better, shows that the RL controllers have an added advantage when the problem is dynamic. In other words, it seems that the performance gain achieved by the controllers is not due to the particular suitability of the EA/problem combination for control but due to the dynamic nature of the specific instances. Unlike the other experiments so far, here, random variation did not have good results: in most cases (seven our of 10), it even had a detrimental effect (including the two static problem instances).

When we take into account the frequency of change in the dynamic problem, we can

[28]It may make sense to change the γ meta-parameter of the controllers that decides the greediness of the learning strategy but we prefer to not introduce another variable factor in our experiments.

Table 6.21: *Results of all experiments. There is a section for each EA including subtables with the results for each test instance. Subtables show the performance of the EA with that problem in terms of online error averaged over all repeats (AOE) and online error of the best runs. In the AOE columns, underlined values are significantly better than static, bold values denote the winner(s) (not significantly worse than the best) and asterisks mark values significantly better than random. Significance is decided by Kolmogorov-Smirnov tests with $\alpha = 0.05$. All problems are minimisation.*

SGA

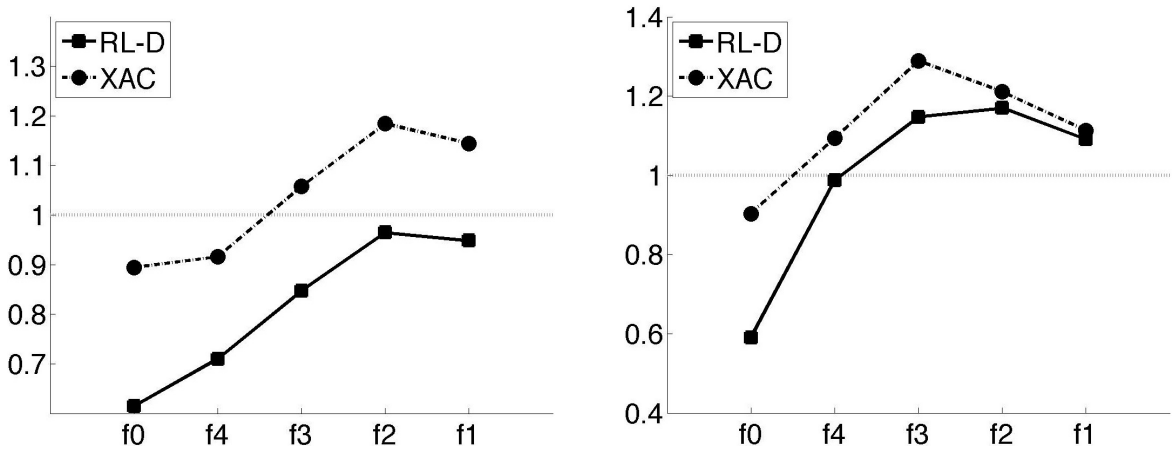
	$-f_0^{DBM}-$		$-f_1^{DBM}-$		$-f_2^{DBM}-$	
	AOE	*Best*	*AOE*	*Best*	*AOE*	*Best*
static	**4129028***	3997601	12545533*	12293306	12818552*	12500407
random	5583692	5297402	14645667	14324007	14879220	14624706
RL-D	6718110	4545744	13228468*	10719852	13287698*	11828774
XAC	4616583*	2546133	**10965472***	8770897	**10825359***	9505985
			$-f_3^{DBM}-$		$-f_4^{DBM}-$	
static			7289058*	7126302	**4935165***	4755202
random			9096672	8787802	6497929	6166802
RL-D			8601781*	6861916	6949285	5571089
XAC			**6890899.27***	5820183	5388935*	4288282

RIGA

	$-f_0^{MPP}-$		$-f_1^{MPP}-$		$-f_2^{MPP}-$	
	AOE	*Best*	*AOE*	*Best*	*AOE*	*Best*
static	**24385827***	6626696	65023926	59051968	66775634	64655237
random	45667364	30563123	59646813	54103325	58191033	54974787
RL-D	41292641	25485233	**59631534**	55569596	57072394*	537 60481
XAC	**26994600***	12049496	**58407227***	55271081	**55140777***	49949091
			$-f_3^{MPP}-$		$-f_4^{MPP}-$	
static			58046717	52860713	43565374*	35492807
random			53453677	48103441	47817810	43300095
RL-D			50590494*	42061776	44125319*	37513821
XAC			**45053562***	38012655	**39832068***	34124232

observe a trend in the performance of the RL controllers, as illustrated in Figure 6.11 where problems are ordered from slower to faster changing. We can see that the achieved performance gain increases as the frequency increases up to a point where it starts dropping again. There may be a good range of change frequency that mostly favours the RL controllers but we must also consider the other dimension of the problems, i.e. the severity of the changes, that also influences the outcomes.

To better understand these results, we look at the behaviour of the controllers; Figure 6.12 shows three of RIGA's parameters over time when solving f_3^{MPP}. What we can first see is that XAC converges around a specific value (for the numeric parameters; the value of the symbolic $xover$ varies). This is contrary to what we would expect considering the



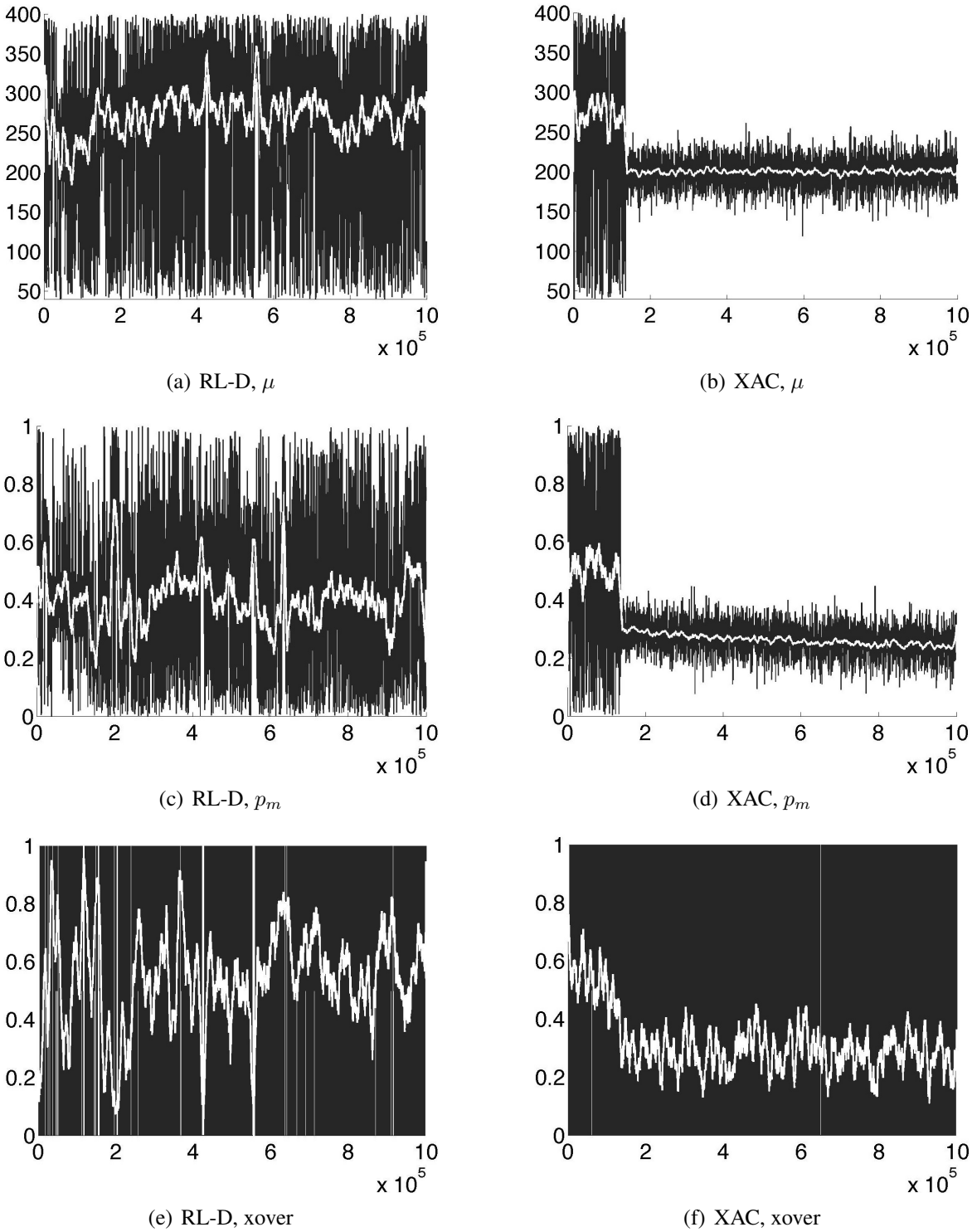
(a) SGA (b) RIGA

Figure 6.11: *The performance gain achieved by RL-D and XAC for all test cases. Gain is calculated as the ratio of performance with static parameters to the performance achieved with the controller. Values higher than 1 denote improvement. Notice that problems are not ordered by index but by their corresponding frequency of change.*

dynamic nature of the problem. Like in the previous section, we run a control experiment to determine if the effect of XAC is to just find a good static value: we run RIGA with static parameters set to the values found by XAC (the average value of the last quarter of XAC's best run with each problem instance).[29] The outcome this time showed that the XAC-tuned static EAs were equally good (no statistically significant difference) as their XAC-controlled counterparts; the XAC controller indeed acted as an online tuner for RIGA.[30] The added advantage with the dynamic problem instances that we mentioned earlier could, in this case, be the amount of training experience available: changes in the environment lead to more training examples as the EA starts a new search and convergence process with each change, making it easier for XAC to learn. On the other hand, RL-D maintains a variation of the parameter values throughout the whole run, however, we cannot determine if that is a reaction to the changing environment or a stochastic effect. As with the other experiments with RL-D so far, we cannot discern any patterns in the parameter values; they appear random. However, the achieved performance is significantly better than that of the random benchmark and the performance gain of RL-D follows the same trend regarding change frequency as discussed earlier. That indicates that RL-D may indeed be able to learn a control policy, though not particularly effective in terms of performance for the SGA.

[29] We did this for all four problem instances.

[30] Even though we saw, in the previous section, that XAC has the capability of finding dynamic policies that



(a) RL-D, μ

(b) XAC, μ

(c) RL-D, p_m

(d) XAC, p_m

(e) RL-D, xover

(f) XAC, xover

Figure 6.12: *Parameter values over time when controlled by RL-D and XAC for RIGA solving the f_3^{MPP} instance. Values shown are from the best runs of each controller for the specific EA/problem setting. The inner white lines are the smoothed curves with a span of 10000.*

6.5.3 Discussion

This section presented the last experiment for this chapter where we have evaluated the two most successful RL controllers with EAs solving dynamic problems. The results were overall inconclusive. Though XAC improved performance for the dynamic problems, it did not do so by dynamic control but by converging to good static values (behaving as an online tuner). On the other hand, RL-D exhibited a more dynamic behaviour, with constantly varying parameter values; the superior performance compared to random variation suggests that it was able to learn meaningful control policies. However, it was able to improve only one of the two EAs.

The analysis of the results also brought forward a previously not considered aspect of parameter control and dynamic environments. The usual argument favouring parameter control in dynamic applications is that, as the environment changes, the optimal parameter values will also change; in other words, the controller should augment the performance of the EA in the dynamic environment. Here we have seen a somewhat reverse effect: dynamic problems frequently refresh the search process and keep the EA in constant movement, thus creating more training experience to be used by the controller; in other words, the dynamic nature of the problem augments the effeciency of the controller.

6.6 Overall Remarks

The experiments and results that were discussed in this chapter form a very big part of the total experimental work of this thesis so here we give some overall remarks. In this chapter we presented four designs of RL based controllers and evaluated their performance in both static and dynamic environments. The two best performing controllers were the discrete RL-D and the continuous XAC. With EAs solving static problems, RL-D had a better success ratio (in terms of improving over using static parameters) but XAC, when effective, was able to achieve better performance gain. With EAs solving dynamic problems, XAC was clearly superior, both in terms of success ratio and performance gain. The hybrid controllers, RL-IP and RL-SMC, performed worse when evaluated with static problems and were not examined further. For all experiments, the meta-parameters of all controllers were set to the same default values that we fixed to values taken from literature or intuitively. This is a positive note regarding the generality and robustness of the controllers and might mean that

react to EA states, it was not able to do so with RIGA.

better informed decisions on these meta-parameters may further improve the controllers' performance. All the RL controllers used a set of observables as a state input; a noteworthy observation is that they performed overall better than the "blind" (i.e. not using a state input) benchmarks.

Despite the fact that we are examining generic controllers that must be able to handle any EA, two evident issues were brought to our attention. First, the monotonicity of the observables of an EA can influence the performance of a controller[31], especially if it is not able to generalise/approximate over observed states. Monotonicity is related to EA features such as restarting and elitism as well as the definition of the observables themselves. Second, the regularity of the EA's search (i.e. how gradual the search progresses) and the number of fitness improvement steps that occur is an influential factor for the function of a controller. Such fitness improvements events constitute the actual experience a controller can learn from; if too sparse, learning can be hindered, though that also depends on the specific controller.

Seeing as all the controllers we worked with are based on reinforcement learning, the definition of the reward used becomes a point of interest. Since there is no standard or previous experience on the matter, we experimented with four different reward definitions (differing in terms of information contained, simplicity and sensitivity) to determine if we could identify a reward type that can work well with different controllers. Results showed that different reward definitions work better for different controllers though differences are not great. In general, the choice of reward will probably not make or break a controller[32] but its influence on performance makes it worth of consideration.

Through observation of the parameter values over time when controlled by XAC and a complimentary experiment, we found evidence supporting that there is indeed benefit in dynamic control versus good static parameter values but that is, however, not necessarily always true and that there can be EA/problem settings were static parameter values can be a better option. Surprisingly, when dealing with dynamic problems, XAC behaved as an online tuner. We cannot determine, though, if the cause for that was XAC's own inadequacy (e.g. an inability to learn an appropriate dynamic policy for the specific EAs/problems or its proneness to premature convergence. Experimentally assessing whether dynamic control or static parameters are optimal for a test case depends on the methods (of control and tuning) used making it a very challenging task.

[31] Making learned experiences unusable if the same situations do not recur in the future,

[32] Of course, as long as it actually represents the learning objective and is not nonsense.

The oddest result of the experiments was the very good performance of random variation of parameter values with EAs solving static problems. Though it did not achieve the performance gain of the better controllers, it still was among the control methods with the best success ratio and it had a detrimental effect on performance (compared to static) in only one case. These statistics are important for two reasons: First, they greatly strengthen the result of Chapter 5 that random variation should always be used as a benchmark when experimentally testing the performance of a parameter controller. Second, random variation can be a very useful tool, e.g. for safely probing the suitability of an EA to parameter control without compromising its performance or as a control method itself when a more sophisticated method is not applicable or available. Contrary to what we would expect, random variation did not yield good results with EAs solving dynamic problems. The reasons for this contradiction are still not clear.

In this chapter we have shown that off-the-shelf generic parameter controllers have promising potential, especially for use with newly developed or ad hoc EAs, and with dynamic environments. Since we are targeting EA-independent control, additional results with more EAs and problems can further improve our understanding of the subject. Moreover, there is ample room for improvement of the presented controllers (we have already identified some points for the XAC controller) and for the development of better designs. Finally, here we have only examined in depth one of the components of a parameter controller, i.e. the algorithm. We believe that it is important to also investigate the definition of observables and of actions.

7

Tuned Parameter Control

LET'S go back to our classification of parameter setting methods with regard to tuning and control in Figure 2.3. The previous chapter covered the lower left box, i.e. control without tuning; this chapter is devoted to the other control enabled category: tuned control. We discuss the concepts related to tuned control, present a design for a tuned controller and experiments that evaluate its performance.

Considering the broad meaning of the word "tuned", we first need to address the question: *what differentiates a tuned controller from an off-the-shelf controller?* Surely, the controllers described in Chapter 6 have several meta-parameters and, thus, can be tuned to a specific EA/problem application at hand. In fact, as has been discussed before, any algorithm will have some parameters and, subsequently, any controller will have some meta-parameters that can be tuned. Though that is true, there is still a difference. An off-the-shelf controller is able to function and learn on the fly without any preparatory calibration while a tuned controller may require an off-line calibration phase in order to function properly. This implies some robustness in the settings of an off-the-shelf controller which is not tuned[1] and creates some additional expectations for the performance of a tuned controller

[1]We refer to the term "tuning" as an algorithmic and systematic search for good parameter vectors as

(to justify the tuning requirement). This differentiation relates to generalist vs specialist discussions regarding solvers [81] and parameter vectors [247]. Additionally, the calibration of a tuned controller can have a much more important aspect than just finding good values for algorithm related parameters: learning good state-to-action mappings. Though a good off-the-shelf controller will use a dynamic mechanism that efficiently learns on-the-fly, a tuned controller may be effective with even a static mechanism that is formed during tuning and takes the necessary actions in certain (previously experienced) states[2]. In that sense not every design from the off-the-shelf control category can transfer to the tuned control class, e.g. the RL controllers from Chapter 6 may have such potential[3] while meta-evolution and PRAM are not particularly fit for the tuned control category.

As with tuning in general, tuned control is particularly fit for repetitive applications for two main reasons. First, in such repetitive settings, the EA is solving a specific problem class making tuning practically effective (as has been previously shown by Smit [247], a parameter vector that is good for one problem might not be that good for another). Second, the accumulated return – in terms of performance gain – over the long term deployment and repeated use of the solver justifies the effort and resources spent for tuning. In the case of tuned control, the controller is added as a "plugin" to the solver to offer the advantages of dynamic parameter values and adaptivity (discussed in Section 2.2 and Chapter 5). The tuning process is applied for the usual reasons but instead of setting the parameter values of the EA (which are now handled by the controller), it calibrates the controller.

The workflow when using tuned control involves two phases as shown in Figure 7.1: an offline calibration phase and a deployment phase. During the calibration phase, a tuning process performs a series of test runs with the combined EA+controller solver. At each run, the controller's meta-parameters (including algorithm parameters and state-to-action mappings) are set to specific values by the tuning process; the result is a utility value produced by the EA that reflects the achieved performance[4]. The tuning process repeats this procedure and finds good values for the meta-parameters of the controller leading to the deployment phase. The settings resulting from the calibration phase are applied to the controller and the solver is ready for use. Since repetitive applications involve long term use, the meta-parameters and state-to-action strategy of the controller need not remain the same after deployment. On

opposed to intuition and literature based selections (such as we made for the off-the-shelf controllers).

[2]This implies that the notion of state might be even more important for tuned controllers.

[3]Notice, however, that certain aspects of those designs are specifically geared towards the off-the-shelf category, e.g. the Temporal Difference learning algorithm and the reward function.

[4]Usually solution quality or speed.

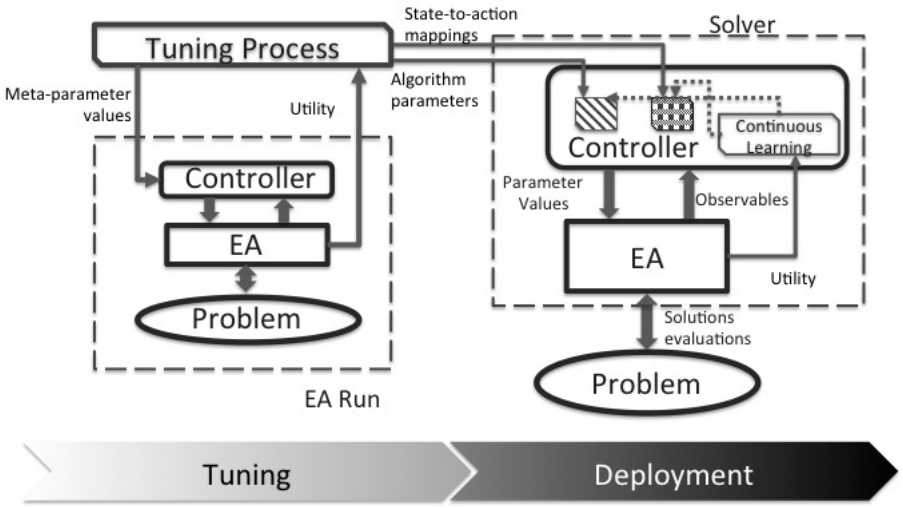


Figure 7.1: *The workflow of tuned control including a temporal perspective.*

the contrary, the experience gathered from recurring solving of the same problem class can be used by the controller for continuous learning[5].

In the following sections we discuss a design for a tuned controller based on Neural Networks (NN) and a model-based parameter tuner used for calibration. We present experiments we carried out to evaluate it's performance and analyse the results.

7.1 A Tuned Controller Based on a Neural Network

In this section we introduce and experimentally evaluate a tuned controller based on a Neural Network (NN), i.e. the controller's mapping mechanism is a feedforward NN taking the observables as inputs and producing the next parameter values as outputs. For the calibration of the controller we used Bonesa, a model-based tuning algorithm [248]. The experiments presented here address the question whether the NN based tuned control approach is viable and what is the added value when compared to tuned static parameter values. Furthermore, we examine the effect of different combinations of observables.

[5]That of course depends on how critical the application is and how much room there is for exploration and "bad" results.

7.1.1 NN Controller Design

First, we discuss the parameters and present a concrete list of observables and subsequently discuss the mapping algorithm and the calibration process.

7.1.1.1 Parameters and Observables

The controller design discussed here is parameter-independent and can handle both numeric and symbolic parameters. Updates occur on every iteration (generation) of the EA. Any numeric parameter can be controlled and the value of a controlled parameter is updated in each generation. To control a parameter, a range $[min, max]$ of values must be specified which will be the output range of the controller.

The observables that we currently use as input to the controller, are based on the current parameter values, diversity and fitness. However, any observable that provides useful information about the current state of the algorithm can be used.

Fitness based We use two observables based on fitness. Both rely on the best fitness digest of the current population: $f_B = max\mathcal{F}(g), g \in G$. The simpler fitness-based observable f_N is the normalised value of the best fitness found in the current population: $f_N = norm(f_B), f_N \in [0, 1]$. This observable is only applicable when the bounds of the fitness function are known. The second fitness-related observable Δf provides information about the change of the best fitness observed in the last generations. Instead of using a derivative to describe this change we instead employ a history of length W and the history function $f_H(f_B^1, ..., f_B^W) = \frac{f_B^W - f_B^{W/2}}{f_B^W - f_B^1}$. This choice of history over a derivative aims at making the controller robust to shifting and stretching of the fitness landscape.

To summarise the two fitness-based observables are:

$$f_N = norm(f_B), f_N \in [0, 1]$$

$$\Delta f = \frac{f_B^W - f_B^{W/2}}{f_B^W - f_B^1}, \Delta f \in [0, 1], W = 100$$

where $f_B = max\mathcal{F}(g), g \in G$. In order to be able to use Δf from the beginning, the value of W grows from 2 to 100 with each generation. Notice that these observables are not used together but are two alternatives.

Diversity based Diversity is observed using the Population Diversity Index (PDI) [249] as the digest function and bypassing derivatives and history. The Population Diversity Index

is a measure that uses an entropy like approach for measuring the genotypic diversity in a real-valued population. It can identify clusters, and always returns a value between 0 and 1 indicating a fully converged (0), a uniformly distributed (1) population, or anything in between.

Current parameter values The current values of the controlled parameters $\vec{p_c}$ are also observed and input to the controller when the Δf observable is used. The reason is that if changes in fitness are observed then changes in the parameter value should be output, thus the old value must be available. Each value in $\vec{p_c}$ corresponds to the current value of a controlled parameter and is normalised to $[0, 1]$ (this is possible because the range of all controlled parameters is known).

Given two choices for observing fitness and the choice to include the diversity observable or not yields four possible sets of observables: $\{f_N\}$, $\{f_N, PDI\}$, $\{\Delta f, \vec{p}\}$ and $\{\Delta f, PDI, \vec{p}\}$. We keep the number of observables to maintain a low number of NN weights that need to be calibrated.

7.1.1.2 Algorithm

As a control method we apply a neural network (NN) as a generic method for mapping real valued inputs to real valued outputs. We use a simple feed-forward network without a hidden layer so as to keep the number of weights lower. The structure of the nodes is fixed and the weights remain static during an EA run. The weights of the NN define the state-to-action mapping; the calibration of the controller is responsible for finding appropriate weights for a given problem or application type. Since we have no labelled training instances[6], conventional supervised learning algorithms cannot be applied. Instead we must perform a search in $\mathbb{R}^n$ for the n weights of the NN. In the subsequent experiment we use a specific tuner but we do not consider it to be an integral choice in the design and could be substituted with another numeric heuristic method.

The inputs to the NN are the observables described above (normalised in the range $[0, 1]$). The weights are tuned in the range $w \in [-1, 1]$. The activation of the neurons is a sigmoid function, horizontally compressed so as to reach very close to its asymptotes given a domain of $[0, 1]$.

When multiple parameters are controlled simultaneously a separate NN controls each parameter. If the current parameter values $\vec{p_c}$ are used as input, then each NN uses only the

[6]Though it would be a very interesting matter to investigate, there is currently no method of discovering what would be the optimal parameter values at a given point of an evolutionary run.

value of the corresponding parameter as input. This may be somewhat oversimplifying considering parameter interaction but it also simplifies the controller and significantly decreases the number of weights that need to be calibrated. Of course, given the capacity to calibrate a much larger number of weights, using a single NN with a hidden layer would be more powerful.

7.1.2 Experimental Setup

We conducted experiments to evaluate the NN-based tuned control approach and how it compares with the tuned static approach. Using the same EA, solving the same problems and spending the same effort for off-line tuning/calibration, we compared the performance achieved when keeping parameter values static to the performance achieved when using the controller. The evolutionary algorithm used, the relevant parameters, the method used for tuning and the overall experimental setup are described in the following subsections.

7.1.2.1 EA and Problems

The EA used, is a Simple ES solving six standard numeric optimisation benchmarks (for details on the algorithm and the problems refer to Appendix B):

- f_1 Sphere
- f_2Corridor
- f_3 Rosenbrock
- f_4 Ackley
- f_5 Rastrigin
- f_6 Fletcher & Powell

We used no recombination and uniform random parent selection, thus leaving three parameters: the population size μ, the generation gap $g = \frac{\lambda}{\mu}$ and the mutation step size σ. We run five distinct experiment sets controlling each parameter alone, controlling μ and g together (considering their obvious interaction) and controlling all parameters together. In the experiments where one of the parameters is not controlled/tuned then it is always set to its default value.

7.1.2.2 Tuning and Overall Setup

Both the calibration of the controller and static values (used for the comparison) are derived through an off-line tuning process using Bonesa [248]. Bonesa is an iterative model-based search procedure based on an intertwined searching and learning loop. The search loop is a generate-and-test procedure that iteratively generates new vectors, pre-assesses their quality using a surrogate model of the performance landscape, and finally tests the most promising vectors by executing an algorithm run with these specific values. In its turn, the learning loop uses the information obtained about the quality of the tested vectors to update the model of the performance surface. Furthermore, it uses a kernel filter to reduce the noise caused by the stochasticity of the algorithm.

For each problem, the four versions of the controller (using the four different observables sets described above) are calibrated using Bonesa. The same tuner, and effort is also spent on finding good static values for the problem, which adheres to classical parameter tuning. The resulting EAs (with controlled and static parameters) are run 100 times to validate the outcomes, to fairly compare their performances. The same experiment is repeated for controlling each of the parameters individually, μ together with g and all together. The experimental setup is summarised in Table 7.1.

Table 7.1: *Experimental Setup*

EA	$(\mu + \lambda)$-ES with: Gaussian mutation, no recombination and uniform random parent selection, limit of 10000 evaluations
Parameters	$\mu \in [0, 1]$ $g \in [0, 14]$ $\sigma \in [1, 100]$ (individually, μ with g and all together)
Instantiation	Bonesa with a budget of 3000 tests to calibrate weights $w_i \in [-1, 1]$
Problems	Sphere, Corridor, Ackley, Rosenbrock, Rastrigin, Fletcher& Powell
Controller observables	$\{f_N\}$, $\{f_N, PDI\}$, $\{\Delta f, \vec{p}\}$, $\{\Delta f$, PDI, $\vec{p}\}$

7.1.3 Results and Analysis

Results are shown in Tables 7.2 to 7.6. For all tables, bold values denote a performance that is not significantly worse than the best on that specific problem, while underlined values denote performance that is significantly better than the EA with static parameter values for

Table 7.2: *Performance results for μ. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	**0.09607**	**9.303**	4.868	**5.75**	36.98	7602
(f_N)	**0.09607**	**9.303**	4.868	**5.75**	36.77	6731
(Δf, D, $\vec{p}$)	**0.09607**	**9.303**	4.868	**5.75**	36.98	6408
(Δf,$\vec{p}$)	**0.09607**	**9.303**	4.868	**5.75**	38.97	5528
Static	**0.09901**	**9.303**	**4.329**	**5.776**	**34.29**	**1335**

Table 7.3: *Performance results for g. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	**0.1009**	**9.303**	**5.115**	**5.716**	**55.15**	1.113e+04
(f_N)	**0.1009**	**9.303**	**5.115**	**5.716**	**55.15**	1.065e+04
(Δf, D, $\vec{p}$)	**0.1009**	**9.303**	6.142	**5.752**	**54.4**	1.065e+04
(Δf,$\vec{p}$)	**0.1009**	**9.303**	**5.115**	**5.716**	**55.32**	1.065e+04
Static	**0.1003**	**9.303**	**5.226**	**5.778**	79.35	**5770**

the specific problem. In all cases, significance is verified using a t-test with 95% confidence. Tuning and solving for the Corridor problem failed in all cases, so it will be completely disregarded in the subsequent analysis.

7.1.3.1 Performance

Tables 7.2, 7.3 and 7.5 show that attempting to control solely the population size, the generation gap or their combination was unsuccessful. But, controlling only σ (Table 7.4) was successful for the controlled EA. It outperforms the static EA on three out of five problems, while not being significantly worse in the other two.

However, controlling all parameters (Table 7.6) at the same time, seems to be the most beneficial; the controlled EA outperforms the static in four problems while it is not significantly worse in the fifth. Even better (although not shown here), on most of the problems, the controller using all parameters outperform those for a single parameter or μ, g combination.

In general, for the EA and problems tested, the controller performs better or at least as good as the tuned, but static parameter values.

Table 7.4: *Performance results for σ. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	0.04848	**9.303**	**7.473**	0.26	29.06	6848
(f_N)	**0.01609**	**9.303**	**6.995**	**0.1085**	**23.32**	**5622**
(Δf, D, $\vec{p}$)	0.0528	**9.303**	**8.05**	**0.3153**	27.08	6089
(Δf,$\vec{p}$)	0.0353	**9.303**	**8.111**	0.617	**25.85**	8238
Static	0.05745	**9.303**	**7.066**	3.684	37.4	**4710**

Table 7.5: *Performance results for μ,g. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	0.09869	**9.303**	6.89	**5.633**	38.26	5841
(f_N)	**0.09382**	**9.303**	4.789	**5.591**	38.66	6432
(Δf, D, $\vec{p}$)	0.09869	**9.303**	6.89	**5.633**	38.19	4299
(Δf,$\vec{p}$)	**0.09382**	**9.303**	4.789	**5.756**	**36.22**	3465
Static	**0.09475**	**9.303**	**3.834**	**5.575**	**34.08**	**762.3**

7.1.3.2 Parameter behaviour

Analysing the behaviour of the controlled parameters provides some important insight.

Controlling μ and g: As the performance results show (Tables 7.2, 7.3 and 7.5) there is no improvement when controlling μ and g. In fact, in most cases, the calibration of control creates controllers that maintain constant parameter values. When controlling μ alone, controllers almost always maintain a constant value, either set to 1 (Sphere, Ackley and Rosenbrock) or simply linearly increasing to its maximum (Rastrigin). Similarly, when controlling g alone, in almost all cases values are kept constant either to its minimum so that each generation has only one offspring (Sphere, Ackley, Rosenbrock) or its maximum (Fletcher & Powell, Rastrigin). In the few cases where parameter values vary accord-

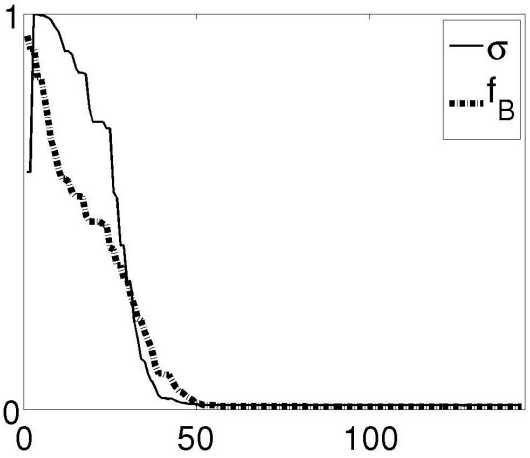


Figure 7.2: *Example of the behavior of parameter σ over time (Ackley problem with $\{f_N\}$). All values are normalized between 0 and 1.*

Table 7.6: *Performance results for all. For each column, bold values denote performance not significantly different to the best and underlined values denote performance significantly better than static.*

	Sph	Crdr	Rsbk	Ack	Rsg	F&P
(f_N,D)	1.102	**9.303**	26.81	3.337	29.28	7824
(f_N)	0.007457	**9.303**	11.54	**0.5488**	**23.99**	5999
(Δf, D, $\vec{p}$)	5.387	**9.303**	82.6	18.5	36.52	2.055e+05
(Δf,$\vec{p}$)	**0.005169**	**9.303**	**7.358**	2.275	30.48	**2028**
Static	0.03565	**9.303**	**7.025**	2.024	35.9	2828

ing to the inputs there is no improvement in performance. These findings could mean that controlling these parameters for the specific EA and problems does not have an intrinsic value or that it is very difficult for the tuning process to perform a good calibration of the controller.

Controlling σ: A much more interesting behaviour is observed in the values of σ. In most cases, σ shows an increase in the beginning and, subsequently, drops and stabilises or oscillates around a value. Such a gradual decrease is a good strategy for climbing a hill and approximating an optimum. An example of this σ behaviour is shown in Figure 7.2. Of course, a preferable alternative would be to re-increase σ when a population is trapped at a local optimum. Such a situation can be detected based on the diversity and Δf observables used, however, this desired behaviour does not occur in our results. A different control strategy is observed in Figure 7.3: σ oscillates rapidly while the range of this oscillation seems related to current diversity. This resembles a random sampling (perhaps with an added "strategy" of an adaptive range) pointing back to the results of Chapter 5.

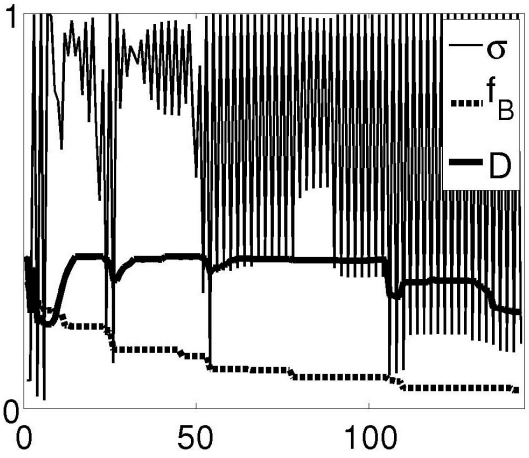
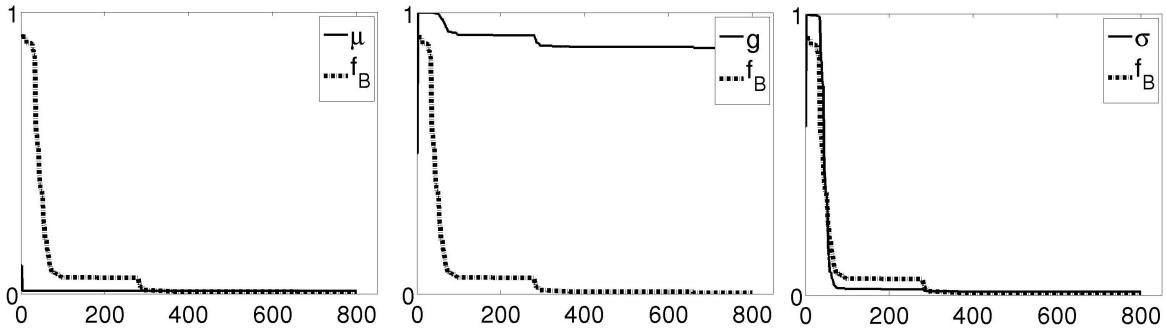


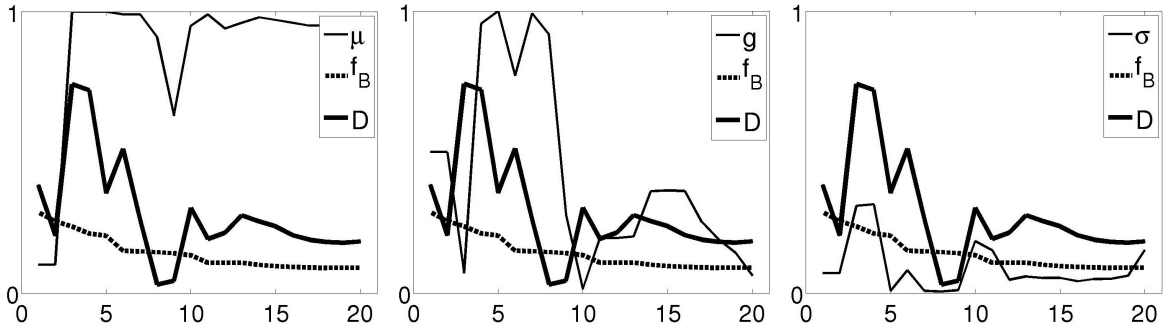
Figure 7.3: *Example of the behaviour of parameter σ over time (Rastrigin problem with $\{\Delta f$, D, $\vec{p}\}$). All values are normalised between 0 and 1.*

Controlling all combined: Combined control of all parameters produces the best results with calibrated controllers generally demonstrating a more complex behaviour. Values of μ and g still often remain constant, especially for the Rosenbrock and Sphere problems (this is not necessarily a drawback considering that these two problems are unimodal and can be



(a) μ (b) g (c) σ

Figure 7.4: *Example of parameter behaviour over time with combined control (Ackley problem with $\{f_N\}$). All values are normalised between 0 and 1.*



(a) μ (b) g (c) σ

Figure 7.5: *Example of parameter behaviour over time with combined control (Rastrigin problem with $\{\Delta f$, D, $\vec{p}\}$). All values are normalised between 0 and 1.*

solved with a $(1+1)$ES). Control of σ shows a behaviour similar to what is described in the previous paragraph. Examples of parameters' behaviour are illustrated in Figures 7.4 and 7.5. In one case μ is set to a constant value, however, g and σ are controlled according to the inputs. In the first case, σ shows one of the two behaviours described in the previous paragraph while in the second it simply follows diversity.

7.1.3.3 Observables

The best results are acquired using only fitness based observables, either $\{f_N\}$ or $\{\Delta f, \vec{p}\}$. Between these two we cannot distinguish a clear winner. Though choosing between f_N or Δf is mostly a matter of feasibility (calculating normalised fitness is not possible if the bounds of the fitness values are not known beforehand), including a diversity observable or not, is a more fundamental question. Contrary to our initial expectations, adding diversity

to the input does not yield better performance even for multimodal problems (including the irregular and asymmetric Fletcher & Powell function). In fact, keeping all other factors fixed, using diversity as an input produces a significant improvement only in 2 out of 50 cases.

7.1.3.4 Discussion on Selection

Three important points were observed in the results and analysis of this section:

- the failure to calibrate a meaningful controller for μ,
- the absence of control strategies that increase σ late in the run to escape local optima,
- and the fact that using diversity as an observable does not improve performance even for multimodal problems.

A plausible assumption is that these points might be due to the survivor selection used by the ES in the experiments ("plus" selection). All above points are related to maintaining a diverse population, either by accommodating enough individuals, by performing enough exploration when necessary or by screening the current status of diversity. However, using a "plus" selection could negate an effort to increase diversity since new and "diverse" individuals would be of inferior fitness and, thus, discarded while newly grown populations would be taken over by the existing best individuals.

7.1.4 Discussion

The experiment in this section shows that tuned control is indeed a viable and fruitful approach that can improve performance compared to using tuned static parameter values. Retaining the same tuning process (and effort) and with a very simple NN as the core of the controller the performance of the EA was significantly improved for most problems.

With respect to the observables chosen, we can conclude that these indeed highly influence the results. While in the previous Chapter we chose observables more "freely", since their number did not really burden the controller, here we had to keep their number low to not increase the difficulty of the calibration process. The good news is that a positive performance gain was achieved by the controller by using only one observable. Remarkably, adding diversity as an input appeared to have hardly any added value for controlling σ.

For one problem, an improved performance was achieved only when all parameters were controlled in combination. Furthermore, it was illustrated that, for certain parameters, the calibration for individual control resulted in a "static strategy", while, when controlled in combination with other parameters, the calibration resulted in an actual varying strategy. These two points suggest that variation and control can highly depend on the interactions between parameters. This was also hinted by the results of the experiment in Chapter 5.

Finally, it is noteworthy that the effect of randomness (see Chapter 5) also appeared here: visualising a successful (in terms of performance) control strategy revealed it was a rapid oscillation, very much resembling a random sampling.

8

Mixing Solver and Parameter Controller

THE entirety of this thesis is devoted to methods of controlling the parameters of an artificial evolutionary process in order to improve its performance. So far all the methods examined and algorithms evaluated approach the matter based on the common assumption that the evolutionary process forms an independent and self-contained algorithm, a (black box) component with inputs and outputs. In that sense, all the control methods presented in the previous chapters were also designed as separate components to be added on the solver: two separate components (EA and controller) communicating and cooperating (through the exchange of observables and parameter values).

This component oriented approach was motivated by the goals of generality and applicability as was explained in Chapter 4: such controllers have wide applicability and can be particularly useful in the adoption of parameter control for real world applications. But, of course, viewing the evolutionary process as a black box component is not a necessary requirement for control in principle: the mechanisms controlling the parameters of the evolution can be injected into the evolutionary algorithm forming an integrated, "mixed" system where evolution is adaptive and the instruments that facilitate this adaptation cannot be disentangled from the whole. Of course, the resulting mixed system will inevitably have some

parameters of its own[1] but that does not matter: the parameters of the *evolutionary process* are adapted and that is what we are interested in here.[2] One of the first forms of control used in EC was in this category: self-adaptation of the mutation step size in Evolution Strategies.

The approach of a mixed evolutionary algorithm and controller is particularly useful in cases where the EA is not a conventional optimiser (implemented as a monolithic centralised algorithm) but, instead, a distributed and, perhaps, physically situated system. Examples of such systems are robot swarms that have to adapt to some dynamically changing environment, or a collection of adaptive software agents that provide services at different locations in a vast computer network. Such systems are becoming more and more important, and so is the need to make them evolvable on-the-fly. The previously introduced controllers would not be applicable to such systems because they (the controllers) are designed for centrally implemented conventional optimisers. In this chapter we present a system called *Fate Agents* that integrates an evolutionary process and the mechanisms for adapting it in a single system and is well suited for such distributed applications. In the second Section of the Chapter we expand the Fate Agents with a simple (external) control method that basically constitutes second order control (since the control of the evolution was already integrated in the original system).

8.1 The Fate Agents System

In traditional EAs we can distinguish two entities: the population of candidate solutions that undergo evolution and a centralised overseer (the main EA loop) that decides about all individuals and performs the evolutionary operators. The system we introduce in this Section decomposes this EA loop into separate functional distributed components that result in a self-regulating system. This system is particularly suitable for evolutionary swarm robotics systems [97],[33],[284] where self-regulation and adaptation are important due to their inherently dynamic circumstances and where a centralised oracle (EA loop) is undesired.[3]

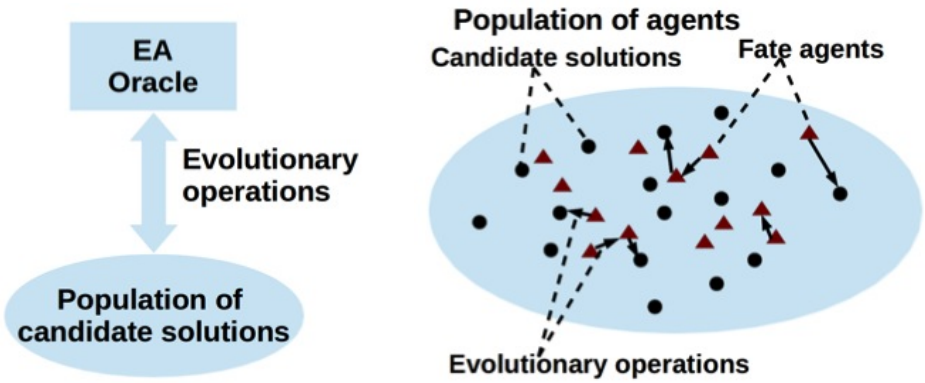
As was mentioned above, the key idea here is to decompose the EA loop into three separate functional components, parent selection, reproduction/variation, and survivor selection, and create autonomous entities that implement these components. We name these entities

[1] As every algorithm or system has.

[2] Notice that the controllers of the previous Chapters also had meta-parameters.

[3] A centralised EA loop forms a single point of failure, limits scalability as it may not be able to process all the information about the individuals in a timely manner, and may not be reachable for certain individuals if the distance exceeds the feasible (or cost effective) range of communication.

Fate Agents after the 'Moirai' of Greek mythology[4]. For a maximally modular system we define three types of Fate Agents, one for each EA component, and add several Fate Agents of each type to the regular population of candidate solutions. These Fate Agents control the evolution of both regular candidate solutions and other Fate Agents in their direct surroundings. Because the Fate Agents also act on Fate Agents, the population of Fate Agents evolves itself and the EA configuration becomes adaptive. Figures 8.1 illustrates the difference between traditional centralised EAs and Fate Agent EAs.



(a) Traditional (b) Fate Agents

Figure 8.1: *Traditional centralised EA, where the evolutionary process is governed by an "EA oracle" contrasted to the Fate Agents EA, where evolutionary operations are executed by independently operating agents.*

Considering the classification of the control method examined here, it is an off-the-shelf control mechanism since it does not require any off-line calibration to function. However, unlike the previous designs presented, it is not a generic controller: though multiple parameters are controlled, the details of the design depend on the specifics of the evolutionary process in the system. Furthermore, even if we consider fate agents just as a concept, only the essential and "universal" parameters of EAs are treated (selection and variation), while for other more specific parameters it is not straightforward how a fate agent could deal with them. In that sense, the control approach examined here is not generic but can be considered more as an ensemble.

The Fate Agents system can be seen as falling in the category of spatially structured EAs [268], where the cellular EA variants are the closest to our system [7]. Nevertheless, there are important differences: spatially structured EAs are based on "oracles" outside the

[4]The three mythological Fates are the incarnations of destiny and control the metaphorical thread of life of every mortal from birth to death.

population that do not change over time, while our Fate Agents operate "from within" and –most importantly– undergo evolution themselves. The combination of spatial structure and parameter control has been studied in [114] and [113], where each location in the grid space has a different combination of parameter values. These, however, are all set by the user at initialisation and do not change over time. The Fate Agents system can also be related to meta-evolution [102], in particular the so-called local meta-evolutionary approaches [239] (not the meta-GA lookalikes). Work in this sub-domain is scarce, we only know about a handful of papers. For instance, [239] provides a theoretical analysis, [151] demonstrates it in GP, while [214] eloquently discusses computational vs. biological perspectives and elaborates on algorithms with an artificial chemistry flavour. The Fate Agents may also be positioned as an evolutionary multi-agent system. In such systems evolution usually facilitates adaptation of the agents, e.g. [71]. However, we are not aware of any evolutionary agent system where the agents actually take control of the evolutionary process itself.

One of the goals of the system presented here is adaptivity in the face of changes in the environment. Evolutionary algorithms have been effectively used in the past for dealing with dynamic environments [37], while such applications have been recognised as one of the main motivations for (self-)adaptive EAs [63]. Dynamism may vary across several dimensions, such as the time between environment changes, change severity or periodicity [57]. Problems range from slowly moving peaks [36] to drastic environment changes that occur after the algorithm has already converged on a solution [39, 293]. Several mechanisms have been proposed to make evolutionary algorithms responsive to environment changes: some generate diversity after a change, while others maintain diversity on the run [149].

The remainder of this Section describes the Fate Agents system in detail and presents an experimental assessment aiming to answer three main questions:

i. Can this evolutionary system solve problems at all?

ii. Can this evolutionary system cope with noise?

iii. Can this evolutionary system cope with changing fitness landscapes?

Because the Fate Agents EA is new and has, to our knowledge, never been implemented before, we are also interested in system behaviour. Therefore, we also inspect various run-time descriptors (e.g., the numbers of agents) that can help us understand what happens during a run.

8.1.1 The Algorithm

Our Fate Agents EA is situated in a (virtual) space where agents move and interact. The evolving population consists of two main types of agents: passive agents that represent candidate solutions to the problem being solved and active Fate Agents that embody EA operators and parameters. Fate Agents form evolving populations themselves because they act not only upon candidate solutions but also upon each other. This makes the Fate Agents EA entirely self-regulated. By design, Fate Agents have a limited range of perception and action: they can only influence other agents within this range. Consequently, the evolutionary process is fully distributed as there is no central authority that orchestrates evolution but different parts of the environment are regulated by different agents. Below we describe the agent types and functionalities and subsequently the algorithm's main cycle.

Candidate Solution Agents

are the simplest type of agent: they only carry a genome which represents a solution to the given problem. The fitness of a candidate solution agent is the fitness of its genome according to this problem. In a swarm robotic application, for example, we could have working robots and Fate robots; the principal problem would then be to evolve controllers for the working robots. The candidate solutions would be the controllers in the working robots encoded by some appropriate data structure and the corresponding fitness would be based on the task the working robots have to solve. To solve an abstract function optimization problem, the candidate solutions' genome would be no different from that in a regular EA, but the candidate solution would be situated in and move about a virtual space, along with the Fate Agents. In general, we assume that candidate solution agents are able to move. However, they are passive in the algorithmic sense, being manipulated by Fate Agents.

Fate Agents

personify and embody evolutionary operators: parent selection, reproduction and survivor selection. Fate Agents have a limited range of operation so that each one can act only within its local neighborhood. Fate Agents themselves form an evolving population, hence they require a measure of fitness. We experimented with various approaches, such as (combinations of) measures like diversity, average and median fitness; we found that the use of the best candidate solution fitness in the area yields the best results. Thus, the fitness of a Fate Agent is set to the fitness of the fittest candidate solution in its neighborhood. There

are three types of Fate Agents, each responsible for different evolutionary operators: *cupids* select and pair up parents, *breeders* create offspring while *reapers* remove agents to make room for new ones. Note that they perform these operations not only on candidate solutions but on each other as well, e.g. cupids make matches between cupids, reapers kill breeders, etc.

Cupids realize parent selection by determining who mates with whom. The selection procedure is the same for all kinds of agents. A cupid creates lists of potential parents by running a series of tournaments in its neighborhood. The number of tournaments held for each type of agent depends on two values: the number of agents of that type in the cupid's neighborhood and a probability that this type of agent is selected. The latter probability is different for each distinct cupid and subject to evolution in the cupid strain. The tournament sizes also evolve. Thus, a cupid's genome consists of four real values representing the selection probabilities for each agent type and one integer for tournament size.

Reapers realise survivor selection indirectly, by selecting who dies. The selection mechanism of reapers is identical to that of cupids (with the difference that reapers' tournaments select the worst of the candidates). Reapers' genomes also consist of four selection probabilities for the different agent types and a tournament size. In earlier versions of the algorithm we tried different mechanisms for cupids and reapers. One approach was allowing a cupid/reaper to examine each and every agent in its neighbourhood and make a separate decision whether to select it or not. That selection decision was facilitated by a simple perceptron using various measures of the agent and its surroundings as input. The weights and the threshold of the perceptron evolved. We found this representation to be overly complicated and results suggested mostly random selection. An initial design of the reapers' selection scheme also included evolving the probability that the winner of a tournament would actually be selected. Results suggested that that probability had no effect (possibly because the size of the tournament already provides sufficient control of selection pressure), thus it was dropped in the final design.

Breeders realise reproduction and variation by producing a child for a given couple of parent agents. For all kinds of agents the breeder performs both recombination and mutation. Breeders, as opposed to cupids and reapers, have different mechanisms for acting upon themselves and upon other agent types. In general, a breeder is given two parents by a cupid in the neighbourhood and applies averaging crossover to produce one offspring and then Gaussian/creep mutation (in our experiments) on that offspring. A breeder's genome consists of three values: the mutation step sizes for candidate solutions, cupids and reapers.

Thus, mutation of these agents evolves in the breeder population. Mutation step sizes for breeders are mutated according to the following rule taken from Evolution Strategies' self-adaptation:

$$\sigma_{t+1} = \sigma_t e^{\tau N(0,1)} \tag{8.1}$$

The reason for this distinction is that if breeders' mutation step sizes were also to evolve then these values would be used to mutate themselves. Trial experiments showed that this approach leads to a positive feedback loop that results in exploding values. Note that the implementation of the breeder depends on the application: the crossover and mutation operators must suit the genomic representation in the candidate solutions. An earlier version of the breeder was designed with the intention to control as much of the reproduction process as possible. The breeders' genome consisted of mutation rates, mutation sizes and different parameters of crossover if applicable. It also included meta-mutation values that were used to mutate the previous values. There were three layers in a breeder's genome: the lower level consisted of values involved in the variation of candidate solutions and other Fate Agents while the upper levels were used to variate the lower layers (and thus the breeders themselves). Results showed that this approach was too complex and inappropriate for evolution, especially since upper level mutation step sizes had a rather minor short-term effect on the fitness of candidate solution agents.

The main cycle

Here we present the general concept of the Fate Agent EA and use abstract function optimisation problems for the design, thus, we had to devise a virtual space and operations for movement. Obviously, applications in a swarm robotics or ALife setting would come with predefined space and movement, and parts of the cycle presented here would be superfluous. All the agents, both candidate solutions and Fate Agents, are situated in the same spatially structured environment, a torus shaped grid. Each cell can either be empty or occupied by exactly one agent. The algorithm makes discrete steps in two phases. The first phase takes care of movement. For this initial design, we have simply used random movement by randomly swapping contents between two neighbouring cells with a certain probability.

In the second phase, evolutionary operators are executed. First, both cupids and reapers make their selections. Subsequently, offspring is produced in iterations as follows: in each iteration, a cupid with available parents and a free cell in its neighbourhood is randomly chosen. A breeder is randomly selected from the neighbourhood of the selected cupid and it is

provided with the parents that are then removed from the cupid's list. The breeder produces a single child which is then placed in the empty cell. This procedure is repeated until there are no cupids with remaining selected agents and unoccupied cells in their neighbourhood.

When offspring production has completed, reaping is performed: reapers are activated in random sequence until there are no reapers left with non-empty selection lists. Notice that during each reaping iteration, a reaper kills only one agent (of any type). Hence, a reaper can kill and be killed in the same reaping iteration. When reaping is complete, the evolutionary phase is concluded and the algorithm starts a new cycle. The overall algorithm cycle is presented in Algorithm 1.

The random sequence and individual actions of cupids and reapers during offspring production and reaping approximate a distributed system with agents acting autonomously and concurrently. It might seem unorthodox that selection by the reapers is performed before offspring are produced, meaning that unfit offspring are allowed to survive and possibly reproduce. Our motivation for this order of operators is to give Fate Agents a 'free pass': before a Fate Agent is considered for removal it should have the chance to act upon its neighbourhood at least once, so that its evaluation will closer reflect its true fitness. The designed order does give this free pass to Fate Agents (at least to cupids and breeders).

8.1.2 Experimental Setup

We conducted several experiments to validate our algorithm from a problem solving perspective and to observe its runtime behaviour. To this end we used the test functions from the BBOB2012 test suite and also the Fletcher & Powell function, because it is very hard to solve and its landscape can be easily redefined for the tests on changing fitness functions (for more information on the test problems refer to Appendix B). We performed three sets of experiments:

i. As a proof-of-concept that the algorithm generally works and is capable of problem solving and self-regulation we used 6 functions: BBOB2012 f_3, f_{20}, f_{22}, f_{23}, f_{24}, and the Fletcher & Powell. We allowed the algorithm to run for 500 generations.

ii. To see if the algorithm can cope with noise we used 6 noisy functions from BBOB2012 (f_{122}, f_{123}, f_{125}, f_{126}, f_{128}, f_{129}) and let the algorithm run for 1000 generations.

iii. To examine how well the system can recover from and adapt to sudden cataclysmic changes we ran tests on the Fletcher & Powell function randomising its matrices every

Algorithm 1 The Fate Agent EA algorithm

```
generation ← 0;
while generation ≤ maxGeneration do
    doMovement;
    for all Cupid c do
        c.SelectParents;
        cupids.Add(c);
    end for
    for all Reaper r do
        r.SelectDeaths;
        reapers.Add(r);
    end for
    while cupids.NotEmpty do
        c ← cupids.GetRandomCupid;
        if c.HasNoFreeCell ∨ c.SelectedParents.Empty then
            cupids.Remove(c);
        else
            b ← c.GetRandomNeighborBreeder;
            cell ← c.GetRandomFreeCell;
            a ← b.Breed(c.GetParents);
            cell.placeAgent(a);
        end if
    end while
    while reapers.NotEmpty do
        r ← reapers.GetRandomReaper;
        if r.agentsToKill.Empty then
            reapers.remove(r);
        else
            r.killAgent;
        end if
    end while
    generation ← generation + 1;
end while
```

250 generations. Here we allowed the algorithm to run for 2000 generations.

For the spatial embedding we used a 100×100 grid where each cell can either be empty or occupied by only one agent (thus there is a maximum of 10000 agents). The grid is initialised by filling all cells with random agents of random type with the following probabilities: 0.0625 for each Fate Agent type and 0.8125 for candidate solutions. Random movement is implemented by swapping the contents of two neighbouring cells with probability 0.5 for each edge. Fate Agents have a neighbourhood of radius 5 cells.

We emphasise that the purpose of these experiments is not to advocate the Fate Agents EA as a competitive numeric optimiser but only to demonstrate its ability to solve problems and investigate its dynamics and self-regulating behaviour without time consuming robotics or ALife experiments. The numeric test suite was used merely as a convenient testbed for evolution, thus a comparison with benchmarks or BBOB champions would be out of context.

8.1.3 Does it work?

The results in section A in Table 8.1 show that the Fate Agents EA is indeed able to solve problems, achieving good fitness on almost all BBOB test functions and a reasonably high fitness for the very difficult FP problem. Good success ratios are also achieved for two noiseless and three noisy functions. Section B of Table 8.1 demonstrates that the system is able to cope with noise very well: it achieves high fitness for all problems and a good success ratio for three out of six noisy functions. As was explained in Section 8.1.2, the purpose of the experiments is not to propose the Fate Agents EA as a numeric optimiser, thus, we will not examine its performance on BBOB any further or determine how competitive it is.

Finally, based on the results of experiment set C, we can give a positive answer to the third question we posed in the beginning of the Chapter: Fig. 8.2 presents the best fitness over time for an example run. The sudden drops in fitness mark the points in time when the matrix of the FP problem is randomly changed, drastically changing the fitness landscape. As can be seen, the system recovers from this catastrophic change, although it does not always succeed. In general, 24 out of 30 runs exhibited at least one successful recovery while, in total, we observed an equal number of successful and unsuccessful recoveries. It should be noted that the FP problem is very hard and the time provided between changes is quite short (250 generations). Nevertheless, results show that the Fate Agent EA does possess the ability to cope with radical change even though the design has no specific provisions for that purpose.

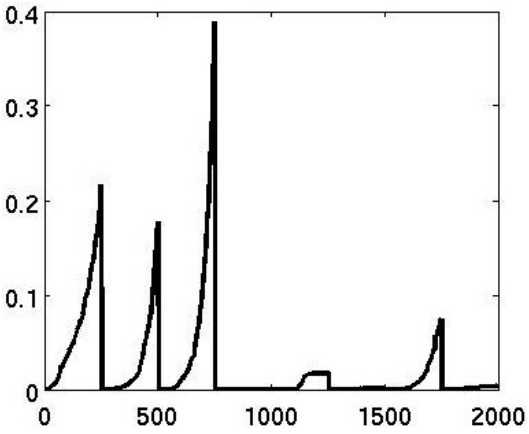


Figure 8.2: *Example of system recovery, fitness vs. time, experiment C, run 4*

Table 8.1: *Performance results for experiment sets A and B in terms of Average Best Fitness normalised between 0 (worst) and 1 (optimal), Average number of Evaluations to Best fitness achieved, Success Rate (success at 0.999 of normalised fitness) and Average number of Evaluations to Success (only successful runs taken into account).*

	Set A - Static Noiseless					*Set B - Static Noisy*			
	ABF	**AEB**	**SR**	**AES**		**ABF**	**AEB**	**SR**	**AES**
F&P	0.53	289682	0.0	-	f_{122}	0.99	207857	0.03	240684
f_3	0.87	223341	0.76	228440	f_{123}	0.99	130996	0.96	131425
f_{20}	0.73	210527	0.03	266631	f_{125}	0.99	131383	0.0	-
f_{22}	0.90	309865	0.8	328891	f_{126}	0.99	126342	1.0	126342
f_{23}	0.90	204382	0.0	-	f_{128}	0.97	172836	0.40	195561
f_{24}	0.05	247797	0.0	-	f_{129}	0.99	137674	0.76	135225

8.1.4 System behavior

One of the most important aspects of the system's behaviour is that the population sizes for the different types of agents are generally successfully regulated. They reach a balance quite different to the initialisation ratios (see Section 8.1.2) very quickly while no agent type becomes extinct or dominates the population even though there are no external limitations imposed.[5] This is a very interesting result on its own right, since regulating population sizes in EAs with autonomous selection is an open issue [286]. An example run is shown in Fig. 8.3. Agent numbers are initialised to default values but populations soon converge and maintain a balance throughout the run. All runs across experiment sets demonstrate similar population dynamics. An adverse attribute of population dynamics that we observed was that, after the population has converged on a solution, the number of reapers is reduced, sometimes to the point of extinction. The reapers eliminate existing individuals and make place for new fate agents and candidate solutions. Without the reapers, after the grid is filled, the whole population is static, and the algorithm cannot respond to changes in the fitness function. In fact, each agent type is crucial for the proper function-

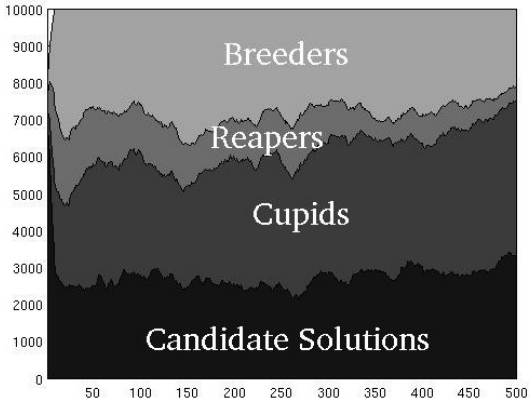


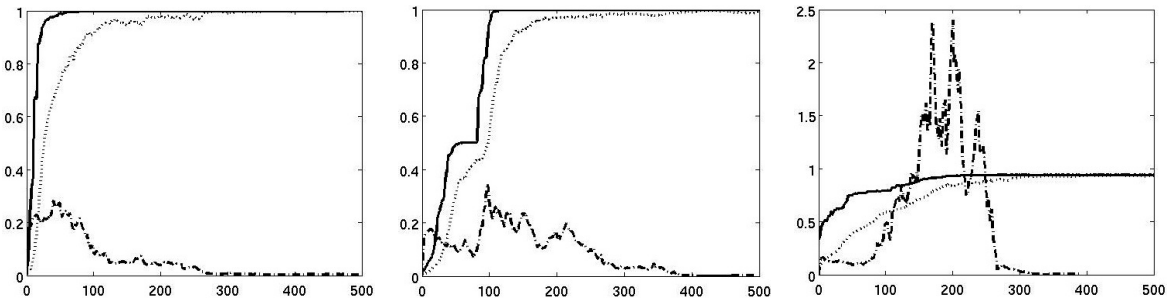
Figure 8.3: *Example of population breakdown over time, experiment A f_{22}, run 12*

[5] We do not make claims regarding optimality here; we are not even aware what an optimal population mix would be (or if it exists).

ing of the algorithm. This effect could be one of the reasons that the system did not always succeed in recovering from landscape changes.

Considering our main focus here, i.e. parameter adaptation, one of the basic expectations is the adaptation of the size of search steps (the mutation step size σ in terms of evolution strategies). In our system, the mutation of agents is controlled by the breeder agents. The breeders' genome includes mutation step sizes for every other agent type. Fig. 8.4 presents examples of three different behaviours observed in breeder populations. Each graph illustrates the best and mean fitness of the candidate solution population and the mutation step size applied to candidate solutions averaged over the whole breeder population. Case (a) is an example of typical evolution with mutation size slowly converging to zero as the search converges to the optimum. Case (b) demonstrates a successful response of the breeders to premature convergence to a local optimum: after around 100 generations the search gets stuck and the breeder population reacts with a steep increase of the mutation size which helps escape and progress. Case (c) shows a failed attempt to escape a local optimum, even though breeders evolve high mutation sizes after the search is stuck.

Note that mutation sizes are correlated to the average fitness, not to the best fitness. This is reasonable considering that Fate Agents have a limited range and are unaware of global values. In all cases, the mutation size converges to zero as soon as the whole population converges (mean fitness becomes equal to the best fitness). This implies that the system has the ability to respond and escape local optima as long as there is still diversity available but is unable to create new diversity after global convergence, as is the case in Fig. 8.4(c).



(a) typical (A, f_{24}, run 10) (b) successful (A, f_3, run 13) (c) failed (A, f_{23}, run 5)

Figure 8.4: *Three examples of breeders' evolution and response to premature convergence. Lines represent best fitness (solid), mean fitness (dotted) and mutation step size for candidate solutions (dashed) over time.*

Finally, we made an interesting observation related to spatial dynamics: results show that, on average, cupids consistently have better fitness than reapers. Both agent types are

evaluated according to the fittest candidate solution in their neighbourhood and both agent types have the same range for these neighbourhoods. We conclude that reapers are usually found in areas with less fit individuals while cupids frequent areas with fitter individuals. Since movement is random, this effect can only be the result of cupids' selection probabilities: cupids in 'bad' areas consistently evolve a preference for selecting reapers while cupids in 'good' areas develop a preference for selecting even more cupids. Furthermore, reapers almost always have very high preference for killing other reapers and, consequently, low ages (they mostly survive only one generation).[6] This implies that cupids in bad areas create 'reaper explosions' that eradicate low-fitness candidate solutions and also clean up after themselves as reapers select each other.

8.1.5 Discussion

The Fate Agents system presented in this section is an evolutionary algorithm that combines solving/optimisation and control of the evolutionary parameters integrated in a single system. This is achieved by decomposing the evolutionary operators and embodying them into active Fate Agents that act on candidate solutions and other Fate Agents. This results in an adaptive and self-regulating system that is particularly fit for distributed applications and dynamic environments.

Experimental results showed that the Fate Agents system exhibits good qualities regarding the control of the evolutionary parameters. The mutation step size is regularly observed to successfully adapt to the situation, e.g. increasing when the population is stuck in a local optimum. Furthermore, the system demonstrates the ability to cope with dynamic (in this case drastically changing) environments. This is a very important point: coping with dynamic environments is one of the main arguments in favour of parameter control (see Section 2.2) and one of the points examined in this thesis (see Chapter 4). The results of these experiments show that adaptation of evolutionary parameters can indeed facilitate such ability.

The Fate Agents system incorporates control of the evolutionary parameters directly into its core algorithm: the mutation step size, the parent selection tournament size etc. are all indirectly controlled through the adaptation of their respective agents. However, when looking at the algorithm as a whole, none of the actual parameters of the system as a unit

[6] These observations are true for almost every run we conducted. Due to lack of space we cannot present relevant graphs.

are controlled. In the following Section, we introduce a simple control mechanism that is added on to the main algorithm as a new component. From a component perspective, it is a typical controller add-on, similar to the fashion of the controllers of the previous Chapters. However, from the perspective of the evolutionary process we can see it as second order meta-control (the first level being the Fate Agents themselves controlling evolutionary parameters).

.

8.2 Adding Meta-Control

One of the main motivations for the Fate Agents system introduced in the previous section is the ability to self-regulate and adapt in dynamic environments.[7] The results of the experiments showed that the system has indeed the potential to cope with drastic changes in the environment (see Section 8.1.3). However, it did not consistently succeed in doing so: only half of the landscape transformations were followed by a successful recovery (in 24 out of 30 runs).

Here we enhance the original Fate Agent system to improve its ability to cope with changes. To this end, we make two adjustments to the algorithm. First, we enforce a limit on the reaper's actions. If the group of agents pertaining to a certain agent type comprises less than 10% of the population in the action range of a reaper, all individuals with that agent type are ignored from killing. This is to prevent the occurrence of reaper extinctions that were sometimes observed after the population converged (see Section 8.1.4) and could obviously pose a problem with adapting to a changing landscape. Second, we redefine the working of the reproduction agents by adding the Adaptive Breeders' Learning Rate (ABLR) mechanism that makes the mutation operator they use adaptive. It is a simple static (stateless), parameter and EA-specific control mechanism aiming to boost exploration whenever an environment change occurs. If we consider that the Fate Agents system already has integrated control of the evolutionary parameters, then the ABLR can be seen as meta-control (though from a strict component perspective it is not).

We assess the enhanced Fate Agents system on three different test scenarios composed

[7]The Fate Agents system is geared towards evolutionary online swarm robotic and multi-agent systems. The need for adaptivity in such systems is obvious, they should be able to adjust their initial pre-deployment settings to the given circumstances that may not be fully known beforehand and they should be able to cope with changes.

of synthetic fitness landscapes that undergo disruptive changes several times during a run. The aim of these tests is to determine wether the new adaptive mechanism helps the Fate Agents EA cope with dynamic environments better and how well it achieves that purpose.

8.2.1 The Adaptive Breeders' Learning Rate

As was explained in the description of the *breeder* agents (see Section 8.1.1), we have avoided allowing breeders to act upon themselves as on other agents. If breeders were to mutate new breeder offspring using the mutation rate encoded in their genomes that would lead to a positive feedback loop that inevitably increases the mutation rates encoded in the breeders' population to very high values. For that reason, in the original algorithm, we made a compromise and performed the mutation of breeders using the standard mechanism employed by Evolution Strategies to mutate their self-adapted strategy parameters.

However, this compromise introduced a static and inflexible component to the otherwise self-regulated Fate Agents system: the population of breeders has a constant rate of mutation, thus a fixed speed of moving around their own meta-level fitness landscape. This drawback is more severe when the problem being solved is dynamic which means that the meta-landscapes of the Fate agents are also changing. In the case of drastic cataclysmic changes in the environment, the candidate solutions have to quickly respond to the new situation by locating and climbing the peaks of the new landscape. This process has to be facilitated by the Fate Agents who, themselves, have to first adapt to the new situation. Both the exploration of the search space by the candidate solutions and the adaptation of the Fate Agents is dependent on the step sizes used to mutate these agents as encoded in the genomes of breeders. These parameters may need to be raised quickly to high values, thus using a static mechanism to mutate them can introduce a sluggishness to the system's response to the change events.

Since all adaptation originates from the breeders, their population is the one that needs to adapt first and fastest. Obviously, a static fixed mutation rate for that is very inappropriate and can greatly slow down the system. For that reason we introduce an the Adaptive Breeders' Learning Rate (ABLR) mechanism that increases the learning rate for mutating breeders (the τ value in Eq. (8.1) of Section 8.1.1) when that is deemed necessary.

The breeder's genome consists of 3 mutation rates: one for the genome of candidate solutions, one for the genome of fate agents, and one for the tournament sizes of cupids and reapers. The learning rate affects all the mutation rates of the new breeders. We are inter-

ested in increasing the diversity in the candidate solutions population, when that is needed, while not directly affecting the diversity of the fate agents population, or the tournament sizes. As such, we will differentiate between the learning rate which is used in the calculation of the breeders' mutation rate for candidate solutions, and the learning rates used in the calculation of the breeders' mutation rates for fate agents' genome and tournament sizes. The former will be manipulated to affect the candidate solutions diversity, whereas the latter will be maintained at a constant level.

Let G be the current generation, and BF_i be the global best fitness at generation i. We assess whether the overall growth (OG)for the last K generations is not positive:

$$OG = BF_G - BF_{G-K+1}, OG \leq 0 \tag{8.2}$$

This can occur when the function is stuck on a local optimum, and no fitness growth is experienced for G generations. More to the interest of this paper, $OG < 0$ when a change in the fitness function decreases the current generation's best fitness. Whenever condition (8.2) is met, we increase the learning rate of the breeders for candidate solutions, τ_{cs}, by a factor LRF:

$$\tau_{cs_{t+1}} = \tau_{cs_t} * LRF$$

The change in the learning rate reflects on an increased candidate solutions' mutation rate (σ_{cs}) of the newly created breeders:

$$\sigma_{cs_{t+1}} = \sigma_{cs_t} e^{\tau_{cs} N(0,1)}$$

We keep increasing the learning rate, until the overall growth becomes positive, at which point the learning rate is reset to its default value. Moreover, the mutation rates for all the breeders are reset to random small values. The reason why we perform this reset is that, after the interval of promoting diversity in the candidate solutions' population, we want to allow the algorithm to converge. The size of the interval over which we measure overall growth affects population diversity and algorithm convergence. Within a small interval, the algorithm may not have time to find a better value that would result in positive overall growth. Consequently, the adaptive learning rate mechanism may be initiated too often, leading to a candidate solutions population with too high diversity and hindering the algorithm's convergence. On the other hand, if the interval over which the overall growth is measured is too big, the learning rate may be increased for an unnecessarily long period: after a change in

the environment growth will be negative until the fitness reaches levels better than what was achieved for the previous environment or, if that is not possible, until at least K iterations have gone by.

As the learning rate increases, many of the breeders will have high mutation rates. Consequently, more of the genes of the candidate solutions may be on the boundaries of their associated interval. As a result, instead of increasing diversity, the proposed mechanism would decrease diversity. To prevent this from happening, whenever, as a result of a mutation, a gene value passes one of its boundaries, it bounces back with a step equal to the extent to which the gene value surpassed the interval. Moreover, the breeders' mutation rates for candidate solutions are limited to the size of the gene interval of candidate solutions.

The ABLR mechanism is a parameter-specific and static controller that uses a history observable based on the best fitness of the population. It has two meta-parameters: the interval K over which growth is measured and the factor LRF used to increase the learning of the breeders when the mechanism is activated.

8.2.2 Experimental Setup

The experiments presented here aim at evaluating the ABLR mechanism and its ability to enhance the performance of the Fate Agents EA when used in dynamic environments. Notice that by "dynamic environment" here we specifically mean landscapes that undergo drastic changes or complete transformations instantly. (As opposed to slowly changing landscapes where the EA needs to track a shifting peak). The goal of these experiments is to answer the following two questions:

- Does the ABLR mechanism improve the performance of the Fate Agents EA in the face of a drastically changing environment?
- How severe is the performance loss when recovering from a change in the environment?

To answer the second question we consider the performance loss relative to a fresh start. That is, we compare the algorithm starting with a converged population after changing the fitness landscape to the same algorithm starting from a random population on the new landscape.

As a test suite for the experiments we used three scenarios A to C where the fitness landscapes undergo complete transformations. Each scenario is divided into five epochs

(e.g. $A_1, A_2, ..., A_5$) with the length of each epoch defined as 250000 evaluations (we use evaluations instead of generations because the Fate Agents EA has a variable population size). During each epoch an entirely different fitness function is used but this function remains the same during a specific epoch. This marks the end of each epoch with a cataclysmic change of the environment and provides four epochs (second to fifth) where the algorithm is evaluated after such a change. The algorithm is informed whenever a change occurs, and a full re-evaluation of all the candidate solutions is performed, to position them in the new fitness landscape. For scenario A, the Fletcher & Powell function was used with five different matrices for the five epochs. For scenarios B and C the BBOB 2013 benchmark suite (see Appendix B for more information) was employed, the functions used in each epoch are shown in Table 8.2.

Table 8.2: *The three scenarios and the corresponding fitness functions used in each of their epochs. Scenario A is based on the Fletcher & Powell function with each epoch using a different matrix. Scenarios B and C use the BBOB benchmark suite, the function numbers used for each epoch are shown in the corresponding order.*

Scenario	Functions	Epochs
A	Fletcher Powell	Five different matrices
B	BBOB	3, 9, 14, 18, 23
C	BBOB	13, 16, 7, 20, 4

We have set the meta-parameters of the ABLR based on preliminary experiments. The interval size K was set to 20 and the learning rate starts with a default value of 0.25, and is increased by a factor of $LRF = 1.25$ (or reset to its default value, whenever the fitness growth becomes positive again).

The original Fate Agents algorithm (FA) and the ABLR enhanced algorithm (FA-ABLR) were run on the above scenarios. Each algorithm and scenario combination was run 40 times with different random seeds to obtain statistically reliable results. All fitness values were normalised in the $[0, 1]$ interval, where 1 represents the problem optimum. We denote the final best fitness achieved at the end of epoch X_i by f_{X_i}. Additionally, separate runs starting with a randomly initialised population were performed with both algorithms solving the individual functions used in all scenarios and epochs. Each of these individual runs were 250000 evaluations long (the same as one epoch of the scenarios). We denote the final best fitness achieved at the end of such an individual run that uses the function of epoch X_i by $f^I_{X_i}$.

8.2.3 Results

In table 8.3 we have compared the performance of the original algorithm and the algorithm with the adaptive breeder learning rate mechanism for the 3 scenarios. The results convincingly demonstrate the advantage of the ABRL method: for all three scenarios, the performance after a change is much higher. The differences can be small (one or two cases), but typically they are large, even one or two orders of magnitude (in eight cases).

To assess the performance loss when the algorithm recovers after a catastrophic change, we have computed the recovery rate by Equation (8.3) that compares the performance of the algorithm with a skewed start (starting on landscape X with a population that converged on landscape Y) to the same algorithm with a fresh start (starting from a random population on landscape X). Formally, it is based on the normalised values of the performance measures, which are averaged over the 40 runs - $\overline{f_{X_i}}$, $\overline{f^I_{X_i}}$.

$$recoveryrate = \frac{\overline{f_X}}{\overline{f^I_X}} \tag{8.3}$$

If performance after a landscape transformation is significantly smaller than performance when starting with random initialisation, recovery rates are also small. However, if performance when starting after a transformation, with a population that converged on a previous function, is the same as the algorithm performance when starting with a randomly initialisation function, then the recovery rate is 1. In this latter case, we can say that the landscape transformation did not incur a performance loss relative to a fresh start. However, the recovery rate may be different from 1, even though the distributions of f_{X_i} and $f^I_{X_i}$ are not significantly different. To properly identify this situations, we have applied a Kolgomorov-Smirnov significance test, with a p-value of 0.05. Table 8.4 presents the recovery rates for the two variants of the algorithm.[8] Given that epoch 1 starts with random initialisation, the calculation of the recovery rate and assessment of performance loss makes sense only for epochs 2-5.

The original algorithm has the worst performance on the Fletcher-Powell function, for which the recovery factor is below 10% for all matrices. The addition of the adaptive breeder's learning rate mechanism improves the recovery rate to the point at which the

[8]Notice that here we are not interested in making a comparison between a full restart and a recovery without restart but, instead, to measure the system's general ability for recovery. To make such measurement we considered that the fitness achieved with a fresh start is, for all practical purposes, the optimal performance this algorithm can achieve with the specific problem and, thus, we use it as a reference point.

Table 8.3: *Comparison of the original Fate Agent EA and the new version using the ABLR. Mean Best Fitness for each scenario/epoch calculated over the 40 runs is shown. All problems are maximisation: larger values are better*

	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
Scenario 1					
Original	0.2285	0.001	0.0255	0.0008	0.0003
ABLR	0.1963	0.1029	0.2371	0.0521	0.0083
Scenario 2					
Original	0.4631	0.1262	0.9994	0.0654	0.5963
ABLR	0.4644	0.183	0.9998	0.6301	0.9172
Scenario 3					
Original	0.6158	0.0596	0.0361	0.298	0.024
ABLR	0.5383	0.5667	0.4394	0.5685	0.1589

Table 8.4: *Comparison of the original Fate Agent EA and the new version using the ABLR. Recovery rates for each scenario/epoch is shown. When the distributions of f_{X_i} and $f^I_{X_i}$ are not significantly different, ˜1 is displayed instead of the calculated recovery rate.*

	Epoch 2	Epoch 3	Epoch 4	Epoch 5
Scenario 1				
Original	0.002	0.0872	0.0074	0.0015
ABLR	˜1	˜1	˜1	˜1
Scenario 2				
Original	0.568	1	0.1	0.626
ABLR	0.838	1	0.888	0.977
Scenario 3				
Original	0.079	0.072	0.475	0.057
ABLR	˜1	0.831	˜1	˜1

performance on a matrix, when the algorithm runs after converging on a previous function, is the same as the performance of the algorithm when it starts off with a random population. ABLR performance is also higher than the original algorithm for the BBOB function sequences, with good recovery rates.

We can get a more clear image of how the adaptive learning rate affects the algorithm by analysing one run of the ABLR algorithm (Fig. 8.5). The observed improvement in performance stems from the fact that, immediately after a function change occurs, the learning rate increases, breeders with higher mutation rates for candidate solutions are created, and

overall diversity is increased. As the new fitness landscape is explored, better solutions are found for the new function, and when the overall growth measure becomes positive, the learning rate is reset. At the same time, the breeders' mutation rate for candidate solutions is reset, leading to a decrease in diversity, and the algorithm converges.

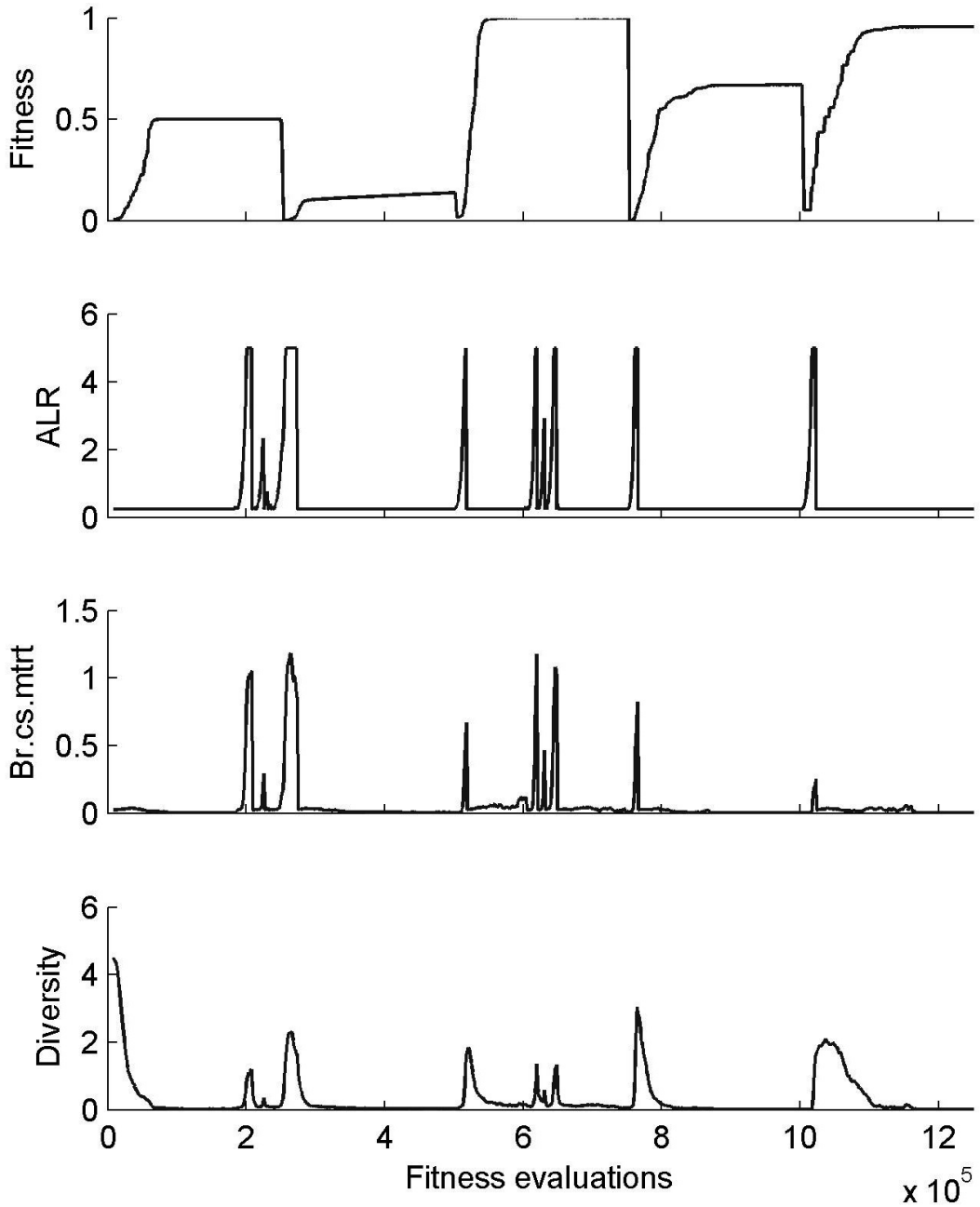
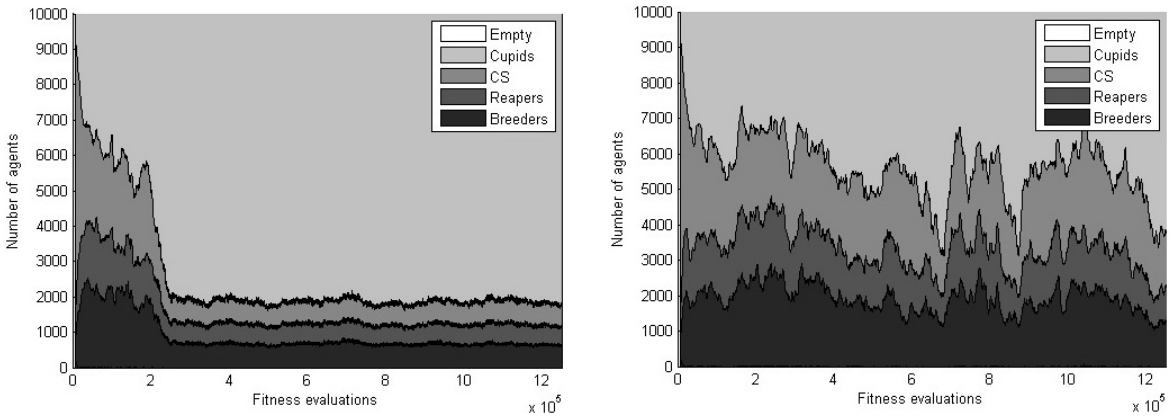


Figure 8.5: *The best fitness, breeder's learning rate for candidate solutions, mutation rate for candidate solutions, and diversity as a measure of entropy(population diversity index [249]), for one run of the scenario 2, with the ABLR algorithm*

The learning rate also increases when the algorithm is stuck on a local optimum, and the overall growth rate is 0, as you can see at the end of section 1 and in the middle of section 3 of the run in Fig. 8.5. We have compared the $f^I_{X_i}$ values for all functions, and found that there is no significant difference between the original and the ABLR algorithm, when the algorithms start with random initialisation. As such, the fact that for the ABLR algorithm learning rate increases when the algorithm is stuck on a local optimum, does not provide an advantage in performance relative to the original algorithm.

8.2.4 Additional insights

Before the current implementation, the reapers population often became completely extinct after the first environment change, so the algorithm could not further proceed in the search. To prevent this we have added a limitation on the reapers, so that they do not kill individuals when the agent type to which they belong has less than 10% individuals in the neighbourhood of the reaper. If we run the original algorithm with this measure in place, after the first change in the environment, the population of reapers, candidate solutions and breeders drops to roughly 10% of the population, while the cupids take over (Fig. 8.6-(a)). With this setup, although the reapers make space for new individuals, most of them are cupids, and just a few are candidate solutions. The first mutations on the converged population may lead to individuals with lower fitness. To maximise fitness, the algorithm protects the current population and minimises change, either by leading the reapers into extinction, or the cupids into taking over.



(a) Original (b) ABLR

Figure 8.6: *Evolution of agent populations for one run of scenario 2, with the original algorithm and with the ABLR mechanism*

This phenomenon does not occur with the ABLR algorithm. After a change occurs, the increase in learning rate forces the algorithm to explore the solution space. In this conditions, an increased population of reapers, candidate solutions, and breeders, is advantageous, and the algorithm is able to restore the population sizes when they are low, and maintain them in balance (Fig. 8.6-(b)).

8.2.5 Discussion

The experimental results provided positive answers to our specific research questions. First, we addressed the question whether the Fate Agents EA with the ABLR mechanism performs better than the original one without ABLR. Results clearly indicated the superiority of the new variant whose maximum fitness values are often orders of magnitude higher than those of the original algorithm. Second, regarding the severity of the performance loss when recovering from a change in the environment, we compared the Fate Agents EA with and without ABLR with a skewed start (starting on landscape X with a population that converged on landscape Y) to the same algorithm with a fresh start (starting from a random population on landscape X). Results showed that the Fate Agents EA with ABLR suffers much less from performance loss (has higher recovery rates) than the original variant. Finally, inspecting the system dynamics during runtime, we have made interesting additional observations. The results showed that the ABLR mechanism also fixes a former behavioural anomaly. Without this mechanism, it occurs that, upon convergence, the populations of breeders, candidate solutions, and reapers drop to around 10%. The ABLR mechanism is able to restore the population levels when a catastrophic change occurs, so that the algorithm can fruitfully explore the new fitness landscape.

These results demonstrate that the simple ABLR control mechanism can greatly improve the ability of the Fate Agents EA to cope with dynamic environments and self-regulate in the event of landscape transformations. This is an important point regarding the value of parameter control in dynamic environments (see Section 2.2). Though the control mechanism is designed specifically for the EA, parameter and purpose (recovering from landscape transformations), it is a clear substantiation of how parameter control can be used to enhance the ability of (already adaptive) EAs to cope with dynamic problems.

9

Concluding Remarks and Future Directions

THIS thesis has attempted a broad study on parameter control for evolutionary algorithms covering many aspects from both a theoretical and a practical perspective. In the first part of the text, we introduced a theoretic framework that positions parameter tuning and control, clarifies their relation and analyses a parameter control mechanism to all its essential components as well as interprets it using reinforcement learning theory. We also provided an extensive survey of the field and observed a trend of increased interest in the field. That interest is, however, concentrated to limited parts and lacks impact; furthermore, we identified challenges in methodology and understanding. In the second part, we used our theoretic framework as a foundation for designing, analysing and assessing a number of control algorithms. We specifically focused on generic (parameter and EA independent) controllers since we believe this is the most promising direction (as was argued in Chapter 4) but also examined the special case of algorithms that fuse optimisation and control of evolutionary parameters into a single system that is designed for a specific application.

9.1 Evaluation of Parameter Control

In the beginning of the second part of this thesis we set out to experimentally assess the following questions:

i. Is an off-the-shelf generic parameter controller a viable idea that can offer benefits in the performance of an EA without the need of calibration to the problem at hand?

ii. Is a tuned generic parameter controller a viable idea that can improve the performance of an EA by calibrating the control policy to suit the problem being solved?

iii. Is there an inherent advantage in having different parameter values at different times during an evolutionary process?

iv. Is parameter control advantageous when facing dynamic problems?

A positive answer to the first question is well supported by the results of Chapter 6. We experimented with several controllers and several EA and problem combinations and observed significant improvement in performance for many of these cases. A consistent trend we observed was that the success of parameter control depended on the competence of the EA with the problem it was solving: the more tailored the EA was to the problem it solved the less the margin of improvement and the less the performance gain achieved with parameter control. A very positive aspect, though, is that the controller also did not worsen the performance of those tailored algorithms (with the exception of one test case). This encourages the vision of a widely applicable generic off-the-shelf controller (perhaps more sophisticated and refined than the control algorithms we presented) that could be applied to any EA/problem offering the potential of significant performance gain without the risk of adverse results.

For tuned control we only performed limited proof-of-concept experiments (Chapter 7); results did show the capacity for improving performance, however, we did not experiment with several EAs, thus we cannot confirm the previously discussed observation regarding the relation between the competence of the EA and the performance gained with control. Furthermore, the control method tested also showed significantly worse results in some cases but that may be a product of the specific (very simple) control algorithm we used. Consequently, at this point we cannot conclude that the ambition for a generic tuned controller is as grounded (as for off-the-shelf control), however, we believe it is a direction that certainly deserves further investigation.

The intrinsic advantage of parameter control and parameter variation itself is harder to evaluate because experimental results with control always depend on the specific control method used. In Chapter 5 we tried to isolate the effect of parameter variation and determine its influence on the performance of an EA. We run an EA with parameter values following several kinds of random variation and compared the resulting performance to that achieved by the EA with optimal (tuned) static parameter values. Results showed that variation by itself can lead to improved performance with some parameters being more important than others; in some cases the interaction between parameters required their concurrent variation for improvements to be made. Considering that the static values were tuned, these results suggest that, for some EA/problem combinations, there is no one single value that can lead to the best performance possible and that different values are better at different times.[1] On the other hand, other results suggest that this may not always be the case: in Section 6.4 we saw that, for the same EA but different problems, in some of the cases the controller dynamically changed the parameters throughout the run but, for other problems, it resorted to converging to a stable value. This may suggest that the same parameter can have different properties when the EA is solving different problems: the same parameter can, in some cases, have a strong attractor (value) or, in others, it should be varied dynamically. More research is needed to confirm if that is true or if it depends on the controller's ability to actually discover a good variation strategy (which may be a more difficult task for some problems than for others).

Finally, regarding the last question, we can say that, in general, the answer is positive. It has been established in previous work that different problems require different parameter settings, thus in the case of fitness landscapes that undergo complete transformations it is obvious that parameter control is beneficial; that was clearly demonstrated in Section 8.2. In the case of problems with moving targets, our findings were less decisive. The results of Section 6.5 showed that dynamic parameter variation improved performance in some cases but it is not always able to do so; additionally, properly set static parameter values achieved even better performance. These results, however, were dependent on the control algorithms used in the experiment. Furthermore, these same results revealed another aspect of the matter: dynamic problems provide a previously overlooked advantage for parameter control: increased training experience that allows a parameter controller to better learn its task.

[1] Of course, this conclusion is based on the assumption that the tuner did a good job in finding the optimal static parameter values.

9.2 When Should I Use Parameter Control?

Based on our research, the most useful guideline we can provide is to use parameter control when working with an EA that is not particularly refined and optimised for the problem at hand, e.g. ad hoc algorithms that are newly developed for solving real world problems. When the EA has been extensively honed to deal with the specific type of problems, the margin of improvement that is available to the controller might be too small. Additionally, dynamic problems are also suitable applications for off-the-shelf parameter control. Whether the controller maintains dynamic parameters or performs online tuning is not important; practically speaking, the attainable performance gain is what matters and not the method by which it is achieved behind the scenes.

We understand that in commercial applications, where the EA's results can have important practical and financial consequences, EA users may hesitate using a parameter controller when there is a risk of getting worse instead of better results. In these cases, we would recommend running a few instances of the EA in parallel with and without control, perhaps in an island model or, even better, using a racing approach that will terminate instances that are clearly worse and avoid wasting function evaluations.

9.3 Notes on Methodology

During our experimental work and analyses we understood that just reporting the performance of a controlled EA and only comparing it to a static version is not enough when evaluating a new controller. There are other aspects and important and useful insights that require more analysis to be seen and understood. First, we believe it is crucial that benchmarks are used when evaluating the performance of a controller, the most important of which is random variation. Regardless of whether the controller improves over the static EA, the comparison to random variation is the essential test that checks if the controller is actually performing a meaningful control or not (in which case it could just be replaced by random variation anyway).

Additionally, further analysis of the behaviour of the controller is necessary. We have experienced first hand that such analysis can some times be far from straightforward, considering the complexity of the methods used and the fact that the underlying controlled evolutionary process is itself poorly understood. However, we suggest that a bare minimum should include visually inspecting how the controlled parameters vary over time; though

there is no guarantee it will show any meaningful patterns, it is still a basic check of the controller's behaviour and it can some times lead to useful observations. Other analyses depend on the specific mechanism used.

In all of our experiments we have tried to make fair comparisons. When comparing algorithms, parameter settings are unavoidably involved in the factors of the experimental setup. It is often the case in literature that the manner by which these settings are decided is not made clear and more effort has gone into finding good parameter values for one algorithm than for another. Since the focus of our work was exactly these parameter settings we made sure that all the participating algorithms in a comparison received the same amount of crafting effort in their setups. In many cases, that amount was zero; of course that means that the results for some algorithms may have been very far from their full potential but that is beyond the point: we think that keeping comparisons fair is important, especially when done in a realistic way that reflects the way in which EAs are used in practice.

9.4 Additional Remarks

Though it was not part of our research objectives, we have experimented with the reward definition used by off-the-shelf controllers to evaluate their actions. These experiments were part of our effort to improve our control algorithms (Sections 6.3 and 6.4) and their results showed that different reward types can be better for different control algorithms, making this is a matter to be considered when designing an off-the-shelf controller. Certain details of these results and of the control algorithms used in the experiments (see Section 6.4.3) provided a hint that some knowledge of past actions and their impact may be important for effective parameter control, however, at this point we cannot safely conclude anything in that direction.

Following the results of Chapter 5 we have used random variation as a baseline benchmark to determine if the improvement of a controller is a result of an intelligent strategy or an effect of the mere variation of parameter values. Nonetheless, in the context of off-the-shelf control, the outcomes of random variation were surprising and impressive (Chapter 6): random parameter variation was often able to improve the performance of EAs (even a competition champion) while it almost never had a adverse effect. Perhaps even more than a benchmark, random variation can be used to quickly check an EA's potential for parameter control or even be used as controller itself if another more sophisticated method cannot be

used. This, however, only applies to our experience with static problems; though we would expect the opposite, random variation did not have the same positive outcomes when dealing with EAs solving dynamic problems.

An important note we need to make here regards EAs themselves rather than the parameter control methods. For the latter to be useful and successful, it is necessary that EA parameters are made external (whether in descriptions/pseudocode in publications or distributed source code) and are clearly explained in terms of range, effect and, possibly, importance (as perceived by their designers or measured by sensitivity tests). This will allow for more reliable and useful results in research and better outcomes in practice.

9.5 Future Directions

When attempting to learn how to decide the best possible course of action given a current situation, it only makes sense that useful and sufficient information about this current situation should actually be available. Just as, in several of our experiments, the progression of values of controlled parameters over time does not seem to follow any intuitive or intelligible pattern but still achieves considerable performance improvement, it would come as no surprise if more effective control could be attained by using observables that are complex, obscure and very different to the simple and intuitive ones that have been, without exception, used so far in the field. Though it is a difficult problem to tackle, we firmly believe that it is a factor with substantial influence on the overall advancement and success of parameter control and deserves focused investigation by itself. An interesting direction would be examining if we can derive observables that maintain the Markov property of the process while still remaining manageable and useful as input to a mapping algorithm; data mining techniques could be used to inspect a sequence of observable values as a time series and evaluate the suitability of these observables for predicting the future. Furthermore, an important question to be answered promptly is whether the best suited set of observables depends on the controller or the EA (or, in the worst case, the combination). Ideally, research on the subject can conclude with a good set of observables to be used by most (or the most employed/successful) controllers or a process that can derive such a good set given a specific controller or a specific EA.

Of course, the above discussion on observables does not imply that the problem of the controller design itself is solved here. On the contrary, the algorithms presented and eval-

uated in this thesis are the initial steps towards the vision of generic and widely applicable parameter controllers. We believe it is important that future research on controller designs moves away from EA-specific methods and follows the direction of generality and applicability discussed in this text. There are several promising ideas to be explored, e.g. examining the settings of the meta-parameters of off-the-shelf controllers and their robustness, kick-starting off-the-self controllers using an auxiliary tuned and robust controller, developing more elaborate designs for tuned controllers that will also be able to learn during deployment (e.g. using Monte Carlo learning) and parallelising EA instances under one control method to allow the controller to better learn and generalise thanks to more and more diverse experience.

Though defining the actions of a control mechanism seems to be a trivial component, there are still a few aspects to be investigated. Here we have used absolute actions, i.e. a controller's action decides a specific value for a parameter. An alternative that deserves investigating is to define controller actions as increments/decrements of the current value. Furthermore, considering that evolution works progressively and improvements can be the result of several steps in the past, an interesting direction would be to develop controllers that can decide on high level composite actions or plans; such an option would be especially appropriate for tuned control.

Finally, we think that much more rapid progress could be achieved if there was standardisation and collaboration similar to that of the EA developing community. The availability of libraries and source codes, the existence of standard EA/problem benchmarks and the organisation of competitions could greatly contribute to the development of better controllers, their analysis and the understanding of which are better and more suitable for which applications.

Appendices

A

Introduction to Reinforcement Learning

This is a short introduction to Reinforcement Learning (RL), mainly the concepts that are used in this thesis. For extensive introductions to RL refer to [263] and [276] while for detailed discussions on more specific subjects in the field of RL refer to [287].

A.1 The RL Problem and Markov Decision Processes

The Reinforcement Learning problem is the problem of learning a decision making strategy through a sequence of interactions with the environment. The setting is shown in Figure A.1: an agent is performing a task within an environment. At time t, the agent can observe the *state* s_t of the environment; based on that state observation, it performs an *action* α_t. Subsequently, the agent receives feedback in the form of a *reward* signal r_t (a scalar value); this reward represents the goal or task of the agent. Also, the action changes the state of the environment into s_{t+1} and the same process is repeated. The purpose of RL is to learn a strategy that maximises the cumulative reward in the long term (called *return*). The return is defined as a *discounted sum* $R = \sum_{t=1}^{N} \gamma^t r_t$, where $0 \leq \gamma \leq 1$ is the *discount rate* that decides how (less) important are future rewards (when $\gamma = 1$ future rewards are equally important

as immediate ones and the return is defined as the simple sum of all received rewards). The goal is to maximise the return; that would mean that the agent performs its task optimally. The form and specification of the states and the actions and the way rewards are calculated depend entirely on the specific problem. Here, we make two important distinctions of RL problems. First, we distinguish between *discrete* problems (finite number of states and actions) and *continuous* problems (states and/or actions come from the space of real numbers). Second, we distinguish between *episodic* tasks and *continuous* tasks. Episodic tasks have a terminal state which leads to resetting to a starting state (e.g. plays of a game or time-limited trials) while continuous tasks are open-ended (no defined ending state).

A RL problem can be formulated as a Markov Decision Process (MDP). A MDP is a tuple $\{S, A, T, R\}$, where S is the set of all states and A is the set of all actions. The transition function T specifies the probability of a state occurring as a result of an action and a previous state $T : S \times A \times S \rightarrow [0, 1]$. The reward function R defines the reward the agent receives given the starting state, the action taken and the resulting state $R : S \times A \times S \rightarrow \Re$. A system is said to have the *Markov property* when the state is defined in such a way that the result of an action only depends on the current state and not the trace of previously visited states, i.e. when

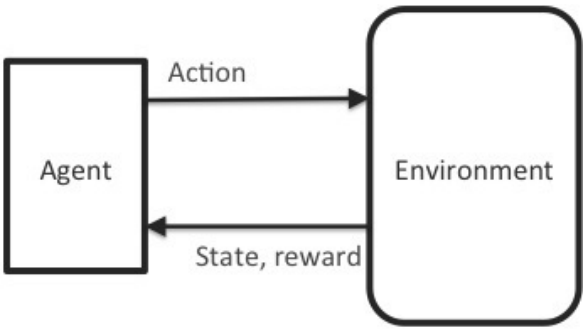


Figure A.1: *The general setting of reinforcement learning.*

$$P(s_{t+1}, r_{t+1}|s_t, a_t) = P(s_{t+1}, r_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) \tag{A.1}$$

A.2 Policy and Value Functions

The decision making strategy that is learned by the RL agent is in the form of a *policy* π. A policy defines a probability that an action is taken in a specific state $\pi : S \times A \rightarrow [0, 1]$ (a special case is a deterministic policy directly defines an action given a state $\pi : S \rightarrow A$). The target of a RL algorithm is learning an optimal policy that maximises the return.

Value functions relate policies and expected returns (and, thus, optimality as well). A *state value function* with regard to policy π, $V^\pi : S \rightarrow \Re$, gives the expected return when

starting with state s and following policy π thereafter:

$$V^{\pi}(s) = E_{\pi}\{\sum \gamma^{k} r_{t+k} | s_t = s\} \tag{A.2}$$

In a similar way, an *action value function* with regard to policy π, $Q^{\pi} : S \times A \rightarrow \Re$, gives the expected return when starting with state s, taking action a and following policy π thereafter:

$$Q^{\pi}(s, \alpha) = E_{\pi}\{\sum \gamma^{k} r_{t+k} | s_t = s, \alpha_t = \alpha\} \tag{A.3}$$

From Eqs A.2 and A.3 we can derive:

$$V^{\pi}(s) = \sum_{\alpha} \pi(s, \alpha) \sum_{s'} T(s, \alpha, s')\{R(s, \alpha, s') + \gamma V^{\pi}(s')\} \tag{A.4}$$

$$Q^{\pi}(s, \alpha) = \sum_{s'} T(s, \alpha, s')\{R(s, \alpha, s') + \gamma max_{\alpha'} Q^{\pi}(s', \alpha')\} \tag{A.5}$$

The *optimal policy* π^{*} that maximises the expected return will also maximise Eqs A.2 and A.3. The Bellman optimality equation (not shown here) shows that, when following π^{*}, the value of a state equals the value of the best action for that state, i.e. being greedy with respect to the value functions is the best approach.

A.3 Methods

If a perfect model of the environment (the T and R functions of the MDP) is known, then the optimal policy can be derived through Dynamic Programming and an incremental process called *policy iteration*. Each step of the policy iteration involves the evaluation of the current policy π (the estimation of V^{π}) and its subsequent improvement. Though for the evaluation of the current policy, a nested iterated process would be required, practically a single update is performed so that the policy evaluation and improvement processes are interleaved. Such an update is achieved with a *backup* operation that replaces the value of a state or action using the current estimates of the successor states. These backups are performed using Eqs A.4 and A.5. Policy improvement is performed by greedily choosing the best known action based on the current estimations of the value functions.

In most real application, however, a perfect model of the environment is not available; the RL algorithms used in these cases are *model-free*: they estimate the value functions using

the state transitions and rewards occurring through interactions with the environment. When dealing with episodic tasks, action values $Q(s, \alpha)$ are calculated as the mean final reward achieved whenever an action α is taken in state s. Monte Carlo (MC) methods approximate these values by starting with a random policy and running several episodes. The final return of each episode is used to update the estimates of the action values and the updated estimates are used to improve the policy.

When facing continuous tasks, Temporal Difference (TD) methods are used that update their value functions and policies after each transition instead of waiting for a terminal state like MC methods. That means that, after each transition, TD methods have to estimate the expected return by using the received reward and the current estimation of the next state (this is based on the discounted sum definition of return):

$$E[R] = r_t + \gamma V(s_{t+1}) \tag{A.6}$$

Because we have to update this estimate after each action, an incremental approach is used to update the state value function:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)) \tag{A.7}$$

where $0 \leq \alpha \leq 1$ is the *step size*. The last part of the formula is called the *TD-error*:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{A.8}$$

One of the main TD algorithms used is SARSA which maintains action values. The updates are similar

$$Q(s_t, \alpha_t) \leftarrow Q(s_t, \alpha_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, \alpha_t)) \tag{A.9}$$

Actor-critic approaches are TD methods that maintain a separate structure representing the policy which is called the *actor*. The *critic* is separate entity that maintains an estimate of the value function and evaluates the actor's choices using the TD-error.

For both MC and TD methods, at any moment, the agent relies on estimations of the value functions that are approximated according to the incomplete knowledge observed so far. It is important that some times the action taken is not the known optimal; this allows a constant exploration and improvement of the approximations of the value functions and the policy by probing the environment in new ways. Exploration is implemented by using ap-

propriate *action selection* schemes, e.g. *softmax* selection assigns to each action a selection probability according to its Q value, while ϵ-*greedy* selects a random action with probability ϵ (otherwise the known best action is selected).

It is often the case with many problems, that the effects of an action only become visible much later (after several other actions have intervened), thus making it impossible for the reward function to directly reflect such effects, e.g. a positive reward r_{t+k} might be the result of action a_t or the combined effect of all actions $(a_t, a_{t+1}, ..., a_{t+k})$. That can be a problem for TD algorithms since they use the immediate rewards to update only the last action taken. To compensate with that problem, *eligibility traces* can be used. In that case, every state-action pair is associated with an eligibility trace $e(s, \alpha)$. At each step, all eligibility traces decay, except for the one of the state-action pair that just occurred that is reset to 1:

$$e_{t+1}(s, \alpha) = \begin{cases} 1 & s = s_t, \alpha = \alpha_t \\ \gamma\lambda e_t(s, \alpha) & otherwise \end{cases} \tag{A.10}$$

where λ is the *decay rate*. Subsequently, the TD-error is applied to the action value function proportionally to the eligibility traces:

$$Q(s, \alpha) \leftarrow Q(s, \alpha) + \alpha\delta_t e_t(s, \alpha), \forall s, \alpha \tag{A.11}$$

B

Methods

B.1 Evolutionary Algorithms

B.1.1 Simple Evolution Strategy (SES)

As the name implies, this is a straightforward implementation of a conventional ES (self-coded). It has real valued representation, Gaussian mutation with one σ and tournament selection for both parents and survivors. The parameters are the population size μ, the generation gap g (the ratio of offspring to population size), the mutation step size σ, the type of survivor selection (plus or comma), the type of crossover (uniform, n-point or arithmetic), the number N of crossover points and the tournament sizes for survivor selection k_s and parent selection k_p. All parameters, their default values and their domains are shown in Table B.1.

Table B.1: *Simple ES parameters, default values and domains.*

Parameter	Default value	Domain
μ	20	$[1, \infty)$
g	2	$(0, \infty)$
σ	0.8	$(0, \infty)$
$Xover$	uni	$\{uni, np, arithm\}$
N	2	$[1, D-1]$
$Selection$	$plus$	$\{plus, comma\}$
k_p	2	$[1, \infty)$
k_s	2	$[1, \infty)$

B.1.2 Cellular GA (CGA)

This Cellular GA [7] was implemented for the BBOB 2013 competition by Holtschulte and Moses [137].[1] Individuals are structured in a two-dimensional toroidal grid with a North-East-West-South neighbourhood. The parameters controlled are the choice of crossover (two-point or arithmetic), the crossover probability p_c, the choice of mutation (Gaussian, uniform, decreasing Gaussian or alternating uniform and Gaussian), the mutation rate p_m and the mutation variance m_{var} (as a percentage of the search domain width). The parent and survivor selection operators are fixed (ranking and select best) and not parameterisable. The size of the grid remains is fixed to 10×10 for all experiments. All parameters of the CGA along with their default values (as found in the original code) and domains are summarised in Table B.2.

Table B.2: *Cellular GA parameters, default values and domains.*

Parameter	Default value	Domain
Xover	$2-point$	$\{2-point, arithm\}$
p_c	0.9	$[0, 1]$
mut	$Gauss$	$\{Gauss, unif, decGauss, alter\}$
p_m	0.1	$[0, 1]$
m_{var}	0.2	$[0, 1]$

[1] The source code was acquired directly from the authors

B.1.3 Genetic Algorithm with Multi-Parent Crossover (GA-MPC)

The GA-MPC by Elsayed et al. [84] was the winner of the CEC 2011 competition on real world applications.[2] It has real-valued representation, multi-parent crossover, and a randomisation operator using an archive. The parameters controlled are the population size μ, the maximum size of the parent tournament k_{max}, the randomisation probability p_r, the percentage of the population that is put in the archive f_a and the mean β_m and standard deviation β_{std} of the normal distribution used to draw the β weight used in the multi-parent crossover. The survivor selection is "select best" and involves no parameters. All the parameters of the GA-MPC with their default values (as they were set in the original source code) and their domains are shown in Table B.3.

Table B.3: *GA-MPC parameters, default values and domains.*

Parameter	Default value	Domain
μ	90	$[1, \infty)$
k_{max}	3	$[3, \infty)$
p_r	0.1	$[0, 1]$
f_a	0.5	$[0, 1]$
β_m	0.7	$[0, \infty)$
β_{std}	0.1	$(0, \infty)$

B.1.4 IPOP-10DDr CMA-ES

The IPOP-10DDr CMA-ES is a CMA ES variant by Liao and Stützle [182] that took part in the BBOB 2013 competition. It is a simple variation of the Increasing Population (IPOP) variant [15]; the difference is that a maximum population size is imposed.[3] The parameters controlled are the backward time horizon for distribution cumulation c_c, the step size cumulation c_s, the step size damping parameter d, the mixing between rank-one and rank-mu update μ_{cov} and the recombination type (equal, linearly decreasing or super-linearly decreasing weights). The population and offspring sizes are controlled at restart by the IPOP-10DDr mechanism and cannot be changed during actual runtime. The restart conditions were kept to defaults. The parameters of the IPOP-10DDr algorithm along with their domains are

[2] At the time of writing the competition web page was online at http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC11-RWP/CEC11-RWP.htm. The source code of GA-MPC was available at the same competition page.

[3] The source code for the (IPOP) CMA-ES was acquired from https://www.lri.fr/~hansen/cmaes_inmatlab.html. The 10DDr variation was added manually.

shown in Table B.4 (the "default" values of the CMA-ES parameters are automatically derived by the algorithm from other parameter values, stopping criteria and the dimensionality of the problem).[4]

Table B.4: *IPOP-10DDr parameters and domains.*

Parameter	Domain
c_c	$[0, 1]$
c_s	$[0, 1]$
d	$[1, 5]$
μ_{cov}	$[1, 20]$
Xover	$\{eq, lindec, superdec\}$

B.1.5 Standard Genetic Algorithm (SGA)

This is a typical generational GA implementation with binary encoding, bit-flip mutation and ranking parent selection. The parameters are the population size μ, the crossover probability p_c, the crossover type (uniform or N-point), the number of crossover points N (used only when crossover is N-point), the mutation probability p_m and the s value used by the ranking process. All the parameters of the SGA with their default values and their domains are shown in Table B.5.

Table B.5: *SGA parameters, default values and domains.*

Parameter	Default value	Domain
μ	100	$[2, \infty]$
p_c	0.8	$[0, 1]$
p_m	$1/D$	$[0, 1]$
Xover	uni	$\{uni, np\}$
N	2	$[1, D-1]$
s	1.5	$[1, 2]$

B.1.6 Random Immigrant Genetic Algorithm (RIGA)

The random immigrant method was suggested by Grefenstette [115] for enhancing a GA to better deal with dynamic problems. RIGA is a real-valued generational GA with Gaussian

[4]The CMA-ES source code has more parameters; here we included all the parameters we were able to change during runtime without interrupting the operation of the algorithm.

mutation with a single step size and tournament parent selection. At each iteration, after the next population is created, the worst r_i fraction of the population is replaced by randomly created individuals. The other parameters are the population size μ, the crossover probability p_c, the type of crossover (uniform or N-point), number of crossover points N (used only when crossover is N-point), the mutation step size p_m and the size of the parent tournament k_p. All the parameters of the RIGA with their default values and their domains are shown in Table B.6.

Table B.6: *RIGA parameters, default values and domains.*

Parameter	Default value	Domain
μ	200	$[1, \infty]$
p_c	0.8	$[0, 1]$
p_m	0.1	$(0, \infty)$
Xover	uni	$\{uni, np\}$
N	2	$[1, D-1]$
k_p	2	$[1, \infty)$
r_i	0.1	[0, 1]

B.2 Problems

B.2.1 Ackley

The Ackley function is a multimodal numeric problem given by

$$f(\vec{x}) = -20 \cdot exp(-0.2 \cdot \sqrt{\frac{1}{D} \cdot \sum_{i=1}^{D} x_i^2} - exp(\frac{1}{D} \cdot \sum_{i=1}^{D} cos(2\pi \cdot x_i)) + 20 + e \qquad \text{(B.1)}$$

where D is the dimensionality of the problem. The search domain is $[-30, 30]$.

B.2.2 Rosenbrock

The Rosenbrock is a unimodal function given by the formula

$$f(\vec{x}) = \sum_{i=1}^{D} (100 \cdot (x_i^2 - x_{i+1})^2 + (1 - x_i)^2) \qquad \text{(B.2)}$$

where D is the dimensionality of the problem. The search domain is $[-2.048, 2.048]$.

B.2.3 Rastrigin

The Rastrigin function is a multimodal numeric optimisation benchmark that exhibits regularity. The objective function is

$$f(\vec{x}) = A \cdot D + \sum_{i=1}^{D} x_i^2 - A \cdot \cos(2\pi x_i) \tag{B.3}$$

where D is the dimensionality of the problem and A is a constant set to 10 for all experiments of this thesis. The search domain is $[-5.12, 5.12]$.

B.2.4 Schwefel

The Schwefel function is a multimodal with regularly distributed local optima numeric optimisation benchmark, given by the formula

$$f(\vec{x}) = 418.9829 \cdot D - \sum_{i=1}^{D} x_i \cdot \sin(\sqrt{|x|}) \tag{B.4}$$

where D is the dimensionality. The search domain is $[-500, 500]$.

B.2.5 Schaffer

The Schaffer problem is multimodal with concentric ridges of equal fitness. The objective function is

$$f(\vec{x}) = \sum_{i=1}^{D-1} (x_i^2 + x_{i+1}^2)^{0.25} \cdot [\sin^2(50 \cdot (x_i^2 + x_{i+1}^2)^{0.1}) + 1] \tag{B.5}$$

where D is the dimensionality of the problem. The search domain is $[-100, 100]$.

B.2.6 Bohachevsky

The Bohachevsky function is a multimodal problem resembling the sphere function but with a "rugged" surface. The objective function is

$$f(\vec{x}) = \sum_{i=1}^{D} (x_i^2 + 2x_{i+1}^2 - 0.3 \cdot \cos 3\pi x_i - 0.4 \cdot \cos 4\pi x_{i+1} + 0.7) \tag{B.6}$$

where D is the dimensionality of the problem. The search domain is $[-50, 50]$.

B.2.7 Griewank

The Griewank function is a multimodal numeric optimisation benchmark with regularly distributed optima. The landscape resembles a "rugged" sphere. the objective function is

$$f(\vec{x}) = \sum_{i=1}^{D} (\frac{x_i^2}{4000}) - \prod_{i=1}^{D} \cos \frac{x_i}{\sqrt{i}} + 1 \tag{B.7}$$

where D is the dimensionality of the problem. The search domain is $[-600, 600]$.

B.2.8 Fletcher & Powell

The Fletcher & Powell function is a difficult benchmark problem with a rugged landscape that is formed by (user-defined or random) parameters. The objective function is

$$f(\vec{x}) = \sum_{i=1}^{D} [\sum_{j=1}^{D} (A_{ij} \cdot \sin \alpha_j + B_{ij} \cdot \cos \alpha_j) - \sum_{j=1}^{D} (A_{ij} \cdot \sin x_j + B_{ij} \cdot \cos x_j)]^2 \tag{B.8}$$

where matrices A, B and vector $\vec{a}$ specify the locations and heights of the extrema. The Fletcher & Powell function is multimodal and asymmetric. Unless otherwise noted, for all experiments in this thesis, we use $A, B, \vec{a}$ as specified in [21].

B.2.9 Shekel

The Shekel function is a difficult benchmark problem that is multimodal and irregular with a few peaks specified by (random or user-defined) parameters. The objective function is

$$f(\vec{x}) = \sum_{i=1}^{m} \frac{1}{c_i + \sum_{j=1}^{D} (x_j - \alpha_{ij})^2} \tag{B.9}$$

where D is the dimensionality of the problem, matrix α defines the locations of the peaks and vector $\vec{c}$ their height. The search domain is $[-10, 10]$.

B.2.10 Black Box Optimisation Benchmark (BBOB) Suite

The BBOB suite is used for the numeric optimisation contest of the same name.[5] The BBOB suite has become one of the standard benchmarks for numeric optimisation; it includes two sets (noiseless and noisy problems) with functions of different characteristics (in terms of separability, modality and symmetry). All experiments in this thesis use the 2013 version of the BBOB suite. For detailed descriptions of the functions we refer the reader to [96].

B.2.11 CEC 2011 Real World Numeric Optimisation Problems

This benchmark set was used for the optimisation competition that was part of the Congress on Evolutionary Computation in 2011.[6] It consists of 22 problems taken from real world applications in domains such as atomic physics, chemistry, sound synthesis, power distribution and spacecraft trajectory optimisation. For detailed descriptions and the objective functions we refer the reader to [59].

B.2.12 Dynamic Bit Matching

This is a dynamic binary problem taken from [261]. The objective function is

$$f(\vec{x}, t) = H(\vec{x}, \vec{T(t)}) \tag{B.10}$$

where $H(\cdot, \cdot)$ is the Hamming distance between two bit strings, and $\vec{T(t)}$ is the target bit string at time t. The target vector t is changed every g evaluations by randomly resetting d randomly selected bits. Parameters N and d are customisable settings that decide the frequency and intensity of change for a problem instance.

[5] At the time of writing, the home page of the BBOB competition was maintained at http://coco.gforge.inria.fr.

[6] At the time of writing, the competition's page was still online at http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC11-RWP/CEC11-RWP.htm.

B.2.13 Moving Peaks

This is a dynamic problem in the real space taken from [35] and involves a landscape with a number of P peaks with changing heights, widths and positions. The objective function is

$$f(\vec{x}, t) = \max_{p=1,\ldots,P} \frac{H_p(t)}{1 + W_p(t) \cdot \sum_{j=1}^{D} (x_j - X_{pj}(t))^2} \tag{B.11}$$

where $H_p(t)$ and $W_p(t)$ are the height and width of peak p respectively at time t and $X_{pj}(t)$ is the position of peak p in the $j-th$ dimension at time t. Every U evaluations, the positions, heights and widths of the peaks are changed as follows

$$H_p(t) = H_p(t-1) + C_H \cdot N(0,1) \tag{B.12}$$

$$W_p(t) = W_p(t-1) + C_W \cdot N(0,1) \tag{B.13}$$

$$\vec{(X)}_p(t) = \vec{(X)}_p(t-1) + \vec{v}_p(t) \tag{B.14}$$

where C_H and C_W are the severities of change for the height and width respectively of all peaks and vector $\vec{v}_p$ is calculated for each peak separately as

$$\vec{v}_p(t) = \frac{s}{|\vec{r} + \vec{v}_p(t-1)|} \cdot \vec{r} \tag{B.15}$$

where s is the severity of the peaks' shift in space and $\vec{r}$ is a random vector in $[-0.5, 0.5]$. The parameters U, C_H, C_W and s determine the frequency and severity of changes in the landscape for a specific problem instance. The domain of the search is [0 100], the ranges of peaks' heights and widths are respectively $[30, 70]$ and $[1, 12]$.

B.3 Controller benchmarks

B.3.1 Random

The random control benchmark simply chooses a value uniformly randomly from the range of control for each parameter independently.

B.3.2 Probabilistic Rule-driven Adaptive Model (PRAM)

The Probabilistic Rule-driven Adaptive Model (PRAM) [289] is one of the few existing generic controllers. PRAM maintains three parameter values and alternates between exploration and exploitation phases. During exploration, these three values are chosen randomly and scores (calculated as fitness improvement) for each one of them are accumulated. At the end of exploration, if the best scoring value was the left or right, the other two move towards it by a fixed step; if the best scoring value was the middle one, the other two move outwards by the same step. During the exploitation phase, values are chosen probabilistically based on the scores of the previous exploration phase. PRAM requires the values of a parameter to be ordered, thus, it can only handle numeric parameters,

B.3.3 Meta-Evolution (ME)

Meta-evolution is implemented using a CMA-ES. The meta-individuals of the CMA-ES are parameter vectors for the controlled EA; each meta-individual is used for one iteration of the controlled EA and the fitness assigned to it is the observed improvement in fitness for the controlled EA. The parameters of the CMA-ES are left to the automatic settings included in the algorithm.[7]

B.4 Statistics

B.4.1 Kolmogorov-Smirnov test

The Kolmogorov-Smirnov (KS) test is used to asses if there are differences between two probability distributions[190]. In specific, we are using the *two-sample* KS test which compares the empirical distributions of two samples. The advantage of the two-sample KS test is that it is based on the empirical distribution functions of the observations and, therefore, does not depend on the underlying distribution functions. The KS tests performed in this thesis use the Matlab implementation[8] that tests the null hypothesis that the two samples are from the same distribution. Rejecting that null hypothesis means that the samples are from different distributions at a 5% significance level.

[7] The source code was acquired from from https://www.lri.fr/~hansen/cmaes_inmatlab.html.
[8] http://nl.mathworks.com/help/stats/kstest2.html

Bibliography

[1] H.A. Abbass. The self-adaptive Pareto differential evolution algorithm. In CEC-2002 [45], pages 831–836.

[2] A. Afanasyeva and M. Buzdalov. Choosing best fitness function with reinforcement learning. In *Proceedings of he 2011 10th International Conference on Machine Learning and Applications*, pages 354–357, 2011.

[3] M. Affenzeller. A new approach to evolutionary computation: Segregative genetic algorithms (SEGA). In *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, volume 2084 of *Lecture Notes in Computer Science*, pages 594–601. Springer Verlag, 2001.

[4] M. Affenzeller. Segregative genetic algorithms (SEGA): A hybrid superstructure upwards compatible to genetic algorithms for retarding premature convergence. *International Journal of Computers, Systems and Signals (IJCSS)*, 2:18–32, 2001.

[5] S. Aine, R. Kumar, and P.P. Chakrabarti. Adaptive parameter control of evolutionary algorithms to improve quality-time trade-off. *Appl. Soft Comput.*, 9(2):527–540, 2009.

[6] E. Alba, H. Ben-Romdhane, S. Krichen, and B. Sarasola. BIPOP: A new algorithm with explicit exploration/exploitation control for dynamic optimization problems. In Yang and Yao [294], pages 171–191.

[7] E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*. Springer, Berlin, Heidelberg, New York, 1st edition, 2008.

[8] A. Aleti. *An Adaptive Approach to Controlling Parameters of Evolutionary Algorithms*. PhD thesis, Swinburne University of Technology, http://users.monash.edu.au/ aldeidaa/papers/thesis.pdf, 2012.

[9] A. Aleti and I. Moser. Predictive parameter control. In Krasnogor et al. [170], pages 561–568.

[10] A. Aleti, I. Moser, and S. Mostaghim. Adaptive range parameter control. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 1–8, Brisbane, Australia, 2012. IEEE Press.

[11] P.J. Angeline. Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[12] J. Arabas, Z. Michalewicz, and J.J. Mulawka. GAVaPS - a genetic algorithm with varying population size. In ICEC-94 [144], pages 73–78.

[13] I. Arnaldo, I. Contreras, D. Milln-Ruiz, J.I. Hidalgo, and N. Krasnogor. Matching island topologies to problem structure in parallel evolutionary algorithms. *Soft Computing*, 17(7):1209–1225, 2013.

[14] D.V. Arnold and A. MacLeod. Step length adaptation on ridge functions. *Evol. Comput.*, 16(2):151–184, 2008.

[15] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In CEC-2005 [46], pages 1769–1776.

[16] A. Auger, M. Schoenauer, and N. Vanhaecke. LS-CMA-ES: A second-order algorithm for covariance matrix adaptation. In Yao *et al* [295], pages 182–191.

[17] A.E. Avramiea, G. Karafotias, and A.E. Eiben. Fate agent evolutionary algorithms with self-adaptive mutation. In *Proceedings of the 2014 Conference Companion on Genetic and Evolutionary Computation Companion*, GECCO Comp '14, pages 191–192. ACM, 2014.

[18] T. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In Männer and Manderick [193], pages 85–94.

[19] T. Bäck. Self-adaptation in genetic algorithms. In F.J Varela and P Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life*, pages 263–271. MIT Press, Cambridge, MA, 1992.

[20] T. Bäck. Parallel optimization of evolutionary algorithms. In Davidor et al. [61], pages 418–427.

[21] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK, 1996.

[22] T. Bäck, A.E. Eiben, and N.A.L. van der Vaart. An empirical study on GAs "without parameters". In Schoenauer *et al* [243], pages 315–324.

[23] T. Bäck and M. Schütz. Intelligent mutation rate control in canonical genetic algorithms. In Z. Ras and M. Michalewicz, editors, *Foundations of Intelligent Systems*, volume 1079 of *Lecture Notes in Computer Science*, pages 158–167. Springer Berlin / Heidelberg, 1996.

[24] P. Bak and K. Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Phys. Rev. Lett.*, 71(24):4083–4086, 1993.

[25] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: An explanation of the $1/f$ noise. *Physical Review Letters*, 59(4):381–384, 1987.

[26] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. Technical Report CMU-CS-95-141, Carnegie Mellon University, Pittsburgh, PA, USA, 1995.

[27] W. Banzhaf *et al*, editor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*. Morgan Kaufmann, San Francisco, 1999.

[28] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*. Springer Verlag, 2008.

[29] J.C. Bean and A. Ben Hadj-Alouane. A dual genetic algorithm for bounded integer programs. Technical Report 92-53, University of Michigan, 1992.

[30] H.G. Beyer and U.M. O'Reilly, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*. ACM, 2005.

[31] H.G. Beyer and B. Sendhoff. Covariance matrix adaptation revisited — the CMSA evolution strategy —. In Rudolph *et al* [233], pages 123–132.

[32] J. Bim, G. Karafotias, S.K. Smit, A.E. Eiben, and E. Haasdijk. It's fate: A self-organising evolutionary algorithm. In C.A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Proceedings of the 12th Conference on Parallel Problem Solving from Nature*, volume 7491–7492 of *Lecture Notes in Computer Science*, pages 185–194. Springer, 2012.

[33] J.C. Bongard. Evolutionary robotics. *Commun. ACM*, 56(8):74–83, 2013.

[34] S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Proceedings of the 11th Conference on Parallel Problem Solving from Nature*, volume 6238–6239 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2010.

[35] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *1999 Congress on Evolutionary Computation (CEC'1999)*, pages 1875–1882. IEEE Press, Piscataway, NJ, 1999.

[36] J. Branke, T. Kaussler, C. Smidt, and H. Schmeck. A multi-population approach to dynamic optimization problems. In Parmee [220], pages 299–307.

[37] J. Branke and S. Kirby. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Boston, 2001.

[38] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657, 2006.

[39] J. Brest, A. Zamuda, B. Boskovic, M.S. Maucec, and V. Zumer. Dynamic optimization using self-adaptive differential evolution. In CEC-2009 [49], pages 415–422.

[40] J. Brest, A. Zamuda, B. Bokovi, S. Greiner, and V. Zumer. An analysis of the control parameters adaptation in DE. In U.K. Chakraborty, editor, *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*, pages 89–110. Springer Berlin Heidelberg, 2008.

[41] L. Budin, M. Golub, and D. Jakobovic. Parallel adaptive genetic algorithm. In *International ICSC/IFAC Symposium on Neural Computation NC98*, pages 157–163, 1998.

[42] A. Buzdalova and M. Buzdalov. Increasing efficiency of evolutionary algorithms by choosing between auxiliary fitness functions with reinforcement learning. In *11th International Conference on Machine Learning and Applications (ICMLA), 2012*, volume 1, pages 150–155, 2012.

[43] E. Cantú-Paz. Migration policies and takeover times in genetic algorithms. In Banzhaf *et al* [27], page 775.

[44] E. Cantú-Paz. Topologies, migration rates, and multi-population parallel genetic algorithms. In Banzhaf *et al* [27], pages 91–98.

[45] *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC'2002)*, Honolulu, USA, 12-17 May 2002. IEEE Press, Piscataway, NJ.

[46] *2005 Congress on Evolutionary Computation (CEC'2005)*, Edinburgh, UK, 2005. IEEE Press, Piscataway, NJ.

[47] *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC'2006)*, Vancouver, Canada, 16-21 July 2006. IEEE Press, Piscataway, NJ.

[48] *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC'2007)*, Singapore, 25-28 September 2007. IEEE Press, Piscataway, NJ.

[49] *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC'2009)*, Trondheim, Norway, 18-21 May 2009. IEEE Press, Piscataway, NJ.

[50] F. Chen, Y. Gao, Z.Q. Chen, and S.F. Chen. SCGA: Controlling genetic algorithms with Sarsa(0). In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 1, pages 1177–1183, 2005.

[51] S.H. Chen, editor. *Evolutionary Computation in Economics and Finance*. Physica-Verlag, 2002.

[52] J. Clune, D. Misevic, C. Ofria, R.E. Lenski, S.F. Elena, and R. Sanjun. Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes. *PLoS Comput Biol*, 4(9):e1000187, 09 2008.

[53] H.G. Cobb. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments. Technical Report 6760 (NLR Memorandum), Naval Research Lab, 1990.

[54] C.A. Coello Coello. Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, 41(2):113–127, 2000.

[55] J.E. Cook and D.R. Tauritz. An exploration into dynamic population sizing. In Pelikan and Branke [222], pages 807–814.

[56] J.C. Costa, R. Tavares, and A. Rosa. An experimental study on dynamic random variation of population size. In *Systems, Man, and Cybernetics, 1999. IEEE International Conference on*, volume 1, pages 607–612, 1999.

[57] C. Cruz, J.R. González, and D.A. Pelta. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15(7):1427–1448, 2011.

[58] L. Da Costa, Á. Fialho, M. Schoenauer, and M. Sebag. Adaptive operator selection with dynamic multi-armed bandits. In Ryan and Keijzer [235], pages 913–920.

[59] S. Das and P.N. Suganthan. Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems. Technical report, Jadavpur University & Nanyang Technological University, 2010.

[60] D. Dasgupta and Z. Michalewicz, editors. *Evolutionary Algorithms in Engineering Applications*. Springer, 1997.

[61] Y Davidor, H.P. Schwefel, and R. Männer, editors. *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, number 866 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, New York, 1994.

[62] K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[63] K.A. De Jong. Parameter setting in EAs: a 30 year perspective. In Lobo et al. [189], pages 1–18.

[64] K.A. De Jong and W.M. Spears. An analysis of the interacting roles of population size and crossover in genetic algorithms. In Schwefel and Männer [246], pages 38–47.

[65] M. de la Maza and B. Tidor. Boltzmann weighted selection improves performance of genetic algorithms. Technical report, MIT A.I. LAB, 1991.

[66] F.F. de Vega, E. Cantú-Paz, J. I. López, and T. Manzano. Saving resources with plagues in genetic algorithms. In Yao *et al* [295], pages 272–281.

[67] C.M. Dinu, P. Dimitrov, B. Weel, and A.E. Eiben. Self-adapting fitness evaluation times for on-line evolution of simulated robots. In C. Blum et al., editors, *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pages 191–198, Amsterdam, The Netherlands, 6-10 July 2013. ACM.

[68] A. Dukkipati, M.N. Murty, and S. Bhatnagar. Cauchy annealing schedule: An annealing schedule for boltzmann selection scheme in evolutionary algorithms. In *Proceedings of the 2004 Congress on Evolutionary Computation : CEC 2004, June 19-23, 2004, Portland, OR, USA*, pages 55–62, Piscataway, NJ, 2004. IEEE.

[69] J. Eggermont, A.E. Eiben, and J.I. van Hemert. A comparison of genetic programming variants for data classification. In D.J. Hand, J.N. Kok, and M.R. Berthold, editors, *Advances in Intelligent Data Analysis, Third International Symposium, IDA-99*, volume 1642 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 1999.

[70] J. Eggermont and J. van Hemert. Adaptive genetic programming applied to new and existing simple regression problems. In J.F. Miller *et al*, editor, *Proceedings of the 4th European Conference on Genetic Programming*, number 2038 in LNCS, pages 23–35. Springer, Berlin, Heidelberg, New York, 2001.

[71] T. Eguchi, K. Hirasawa, J. Hu, and N. Ota. A study of evolutionary multiagent models based on symbiosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(1):179–193, 2006.

[72] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[73] A.E. Eiben, M. Horvath, W. Kowalczyk, and M.C. Schut. Reinforcement learning for online control of evolutionary algorithms. In Brueckner, Hassas, Jelasity, and Yamins, editors, *Proceedings of the 4th International Workshop on Engineering Self-Organizing Applications (ESOA'06)*, volume 4335, pages 151–160. Springer, 2006.

[74] A.E. Eiben and M. Jelasity. A critical note on Experimental Research Methodology in EC. In CEC-2002 [45], pages 582–587.

[75] A.E. Eiben, E. Marchiori, and V.A. Valko. Evolutionary algorithms with on-the-fly population size adjustment. In Yao *et al* [295], pages 41–50.

[76] A.E. Eiben, Z. Michalewicz, M. Schoenauer, and J.E. Smith. Parameter control in evolutionary algorithms. In Lobo et al. [189], pages 19–46.

[77] A.E. Eiben and Z. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *International Conference on Evolutionary Computation*, pages 258–261, 1996.

[78] A.E. Eiben and A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998.

[79] A.E. Eiben, M.C. Schut, and A.R. de Wilde. Is self-adaptation of selection pressure and population size possible? - a case study. In PPSN-2006 [234], pages 900–909.

[80] A.E. Eiben and S.K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[81] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg, 2003.

[82] A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[83] A.E. Eiben and J.I. van Hemert. SAW-ing EAs: Adapting the fitness function for solving constrained problems. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, Advanced Topics in Computing, chapter 26, pages 389–402. McGraw-Hill, 1999.

[84] S.M. Elsayed, R.A. Sarker, and D.L. Essam. GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems. In *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*, pages 1034–1040, New Orleans, USA, 2011. IEEE Press.

[85] R. Farmani and J.A. Wright. Self-adaptive fitness formulation for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 7(5):445–455, 2003.

[86] C. Fernandes and A. Rosa. A study on non-random mating and varying population size in genetic algorithms using a royal road function. In *2001 Congress on Evolutionary Computation (CEC'2001)*, pages 60–66. IEEE Press, Piscataway, NJ, 2001.

[87] C. Fernandes and A. Rosa. Self-regulated population size in evolutionary algorithms. In Runarsson *et al* [234], pages 920–929.

[88] C. Fernandes, R. Tavares, and A.C. Rosa. niGAVaPS - outbreeding in genetic algorithms. In *Proceedings of the 2000 ACM symposium on Applied computing - Volume 1*, SAC '00, pages 477–482. ACM, 2000.

[89] C.M. Fernandes, J.L.J. Laredo, A.M. Mora, A.C. Rosa, and J.J. Merelo. The sandpile mutation operator for genetic algorithms. In *Proceedings of the 5th international conference on Learning and Intelligent Optimization*, LION'05, pages 552–566. Springer-Verlag, 2011.

[90] C.M. Fernandes, J.L.J. Laredo, A.M. Mora, A.C. Rosa, and J.J. Merelo. A study on the mutation rates of a genetic algorithm interacting with a sandpile. In *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*, EvoApplications'11, pages 32–42. Springer-Verlag, 2011.

[91] C.M. Fernandes, J.J. Merelo, V. Ramos, and A.C. Rosa. A self-organized criticality mutation operator for dynamic optimization problems. In Ryan and Keijzer [235], pages 937–944.

[92] F. Fernandez, M. Tomassini, and L. Vanneschi. Saving computational effort in genetic programming by means of plagues. In *2003 Congress on Evolutionary Computation (CEC'2003)*, pages 2042–2049, Edinburgh, UK, 2003. IEEE Press, Piscataway, NJ.

[93] Á. Fialho. *Adaptive Operator Selection for Optimization*. PhD thesis, Universite Paris-Sud, http://tel.archives-ouvertes.fr/docs/00/57/84/31/PDF/thesisAlvaro.pdf, 2010.

[94] Á. Fialho, L. Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In Rudolph *et al* [233], pages 175–184.

[95] Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1-2):25–64, 2010.

[96] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2010: Presentation of the noiseless functions. Technical report, 2013.

[97] D. Floreano, P. Husbands, and S. Nolfi. Evolutionary robotics. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 1423–1451. Springer Berlin Heidelberg, 2008.

[98] D.B. Fogel, L.J. Fogel, and J.W. Atmar. Meta-evolutionary programming. In *Signals, Systems and Computers, 1991. 1991 Conference Record of the Twenty-Fifth Asilomar Conference on*, pages 540 –545 vol.1, 1991.

[99] G.B. Fogel and D.W. Corne, editors. *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann, 2002.

[100] S. Forrest, editor. *Proceedings of the 5th International Conference on Genetic Algorithms*. Morgan Kaufmann, San Francisco, 1993.

[101] G. Francesca, P. Pellegrini, T. Stützle, and M. Birattari. Off-line and on-line tuning: A study on operator selection for a memetic algorithm applied to the qap. In P. Merz and J.K. Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6622 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2011.

[102] B. Freisleben. Meta-evolutionary approaches. In T Bäck, D.B Fogel, and Z Michalewicz, editors, *Handbook of Evolutionary Computation*, pages 214–223. Institute of Physics Publishing, Bristol, and Oxford University Press, New York, 1997.

[103] B. Freisleben and M. Härtfelder. Optimization of genetic algorithms by genetic algorithms. In RudolfF. Albrecht, ColinR. Reeves, and NigelC. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 392–399. Springer Vienna, 1993.

[104] R. Gämperle, S.D. Müller, and P. Koumoutsakos. A parameter study for differential evolution. In *WSEAS Int. Conf. on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298, 2002.

[105] A. Ghosh, A. Chowdhury, R. Giri, S. Das, and S. Das. A fitness-based adaptation scheme for control parameters in differential evolution. In Pelikan and Branke [222], pages 2075–2076.

[106] A. Ghosh, S. Das, A. Chowdhury, and R. Giri. An improved differential evolution algorithm with fitness-based adaptation of the control parameters. *Information Sciences*, 181(18):3749 – 3765, 2011.

[107] D.E. Goldberg. A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4:445–460, 1990.

[108] D.E. Goldberg, K. Deb, and J.H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6(4):333–362, 1992.

[109] D.E. Goldberg, K. Deb, and J.H. Clark. Accounting for noise in the sizing of populations. In L.D Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 127–140. Morgan Kaufmann, San Francisco, 1993.

[110] D.E. Goldberg, K. Sastry, and T. Latoza. On the supply of building blocks. In Spector *et al* [259], pages 336–342.

[111] J. Gomez. Self adaptation of operator rates in evolutionary algorithms. In Kalyanmoy Deb *et al*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, volume 3102 of *Lecture Notes in Computer Science*, pages 1162–1173. Springer, 2004.

[112] W. Gong, Á. Fialho, and Z. Cai. Adaptive strategy selection in differential evolution. In Pelikan and Branke [222], pages 409–416.

[113] Y. Gong and A. Fukunaga. Distributed island-model genetic algorithms using heterogeneous parameter settings. In *IEEE Congress on Evolutionary Computation*, pages 820–827, 2011.

[114] V.S. Gordon, R. Pirie, A. Wachter, and S. Sharp. Terrain-based genetic algorithm (TBGA): Modeling parameter space as terrain. In Banzhaf *et al* [27], pages 229–235.

[115] J. Grefenstette. Genetic algorithms for changing environments. In Männer and Manderick [193], pages 137–144.

[116] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122 –128, 1986.

[117] E. Haasdijk, S.K. Smit, and A.E. Eiben. Exploratory analysis of an on-line evolutionary algorithm in simulated robots. *Evolutionary Intelligence*, 5(4):213–230, 2012.

[118] S. B. Hamida and M. Schoenauer. An adaptive algorithm for constrained optimization problems. In Schoenauer *et al* [243], pages 529–538.

[119] N. Hansen. The CMA evolution strategy: a comparing review. In J.A Lozano and P. Larranaga, editors, *Towards a New Evolutionary Computation : Advances in Estimation of Distribution Algorithms*, pages 75–102. Springer, Berlin, Heidelberg, New York, 2006.

[120] N. Hansen. Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2389–2396. ACM, 2009.

[121] N. Hansen. Benchmarking a BI-population CMA-ES on the BBOB-2009 noisy testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2397–2402. ACM, 2009.

[122] N. Hansen and S. Kern. Evaluating the cma evolution strategy on multimodal test functions. In Yao *et al* [295], pages 282–291.

[123] N. Hansen, S.D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18, 2003.

[124] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In ICEC-96 [145], pages 312 –317.

[125] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[126] N. Hansen and R. Ros. Benchmarking a weighted negative covariance matrix update on the BBOB-2010 noiseless testbed. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, GECCO '10, pages 1673–1680. ACM, 2010.

[127] G.R. Harik, E. Cantú-Paz, D.E. Goldberg, and B.L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evol. Comput.*, 7(3):231–253, 1999.

[128] G.R. Harik and F.G. Lobo. A parameter-less genetic algorithm. In Banzhaf *et al* [27], pages 258–265.

[129] G.R. Harik, F.G. Lobo, and D.E. Goldberg. The compact genetic algorithm. *Evolutionary Computation, IEEE Transactions on*, 3(4):287 –297, 1999.

[130] M. Harman. Software engineering meets evolutionary computation. *Computer*, 44(10):31–39, 2011.

[131] F. Herrera and M. Lozano. Adaptation of genetic algorithm parameters based on fuzzy logic controllers. In *Genetic Algorithms and Soft Computing*, pages 95–125. Physica-Verlag, Heidelberg, Germany, 1996.

[132] J. Hesser and R. Männer. Towards an optimal mutation probability for genetic algorithms. In Schwefel and Männer [246], pages 23–32.

[133] R. Hinterding. Gaussian mutation and self-adaption for numeric genetic algorithms. In *Proceedings of the 1995 IEEE Conference on Evolutionary Computation*, pages 384–389. IEEE Press, Piscataway, NJ, 1995.

[134] R. Hinterding, Z. Michalewicz, and T. Peachey. Self-adaptive genetic algorithm for numeric functions. In H.-M Voigt, W Ebeling, I Rechenberg, and H.-P Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, number 1141 in Lecture Notes in Computer Science, pages 420–429. Springer, Berlin, Heidelberg, New York, 1996.

[135] T. Hiroyasu, M. Miki, and M. Negami. Distributed genetic algorithms with randomized migration rate. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 1, pages 689 –694, 1999.

[136] J.H Holland. *Adaption in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 1992. 1st edition: 1975, The University of Michigan Press, Ann Arbor.

[137] N.J. Holtschulte and M. Moses. Benchmarking cellular genetic algorithms on the BBOB noiseless testbed. In *Proceeding of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion*, GECCO '13 Companion, pages 1201–1208. ACM, 2013.

[138] T. Hu, S. Harding, and W. Banzhaf. Variable population size and evolution acceleration: a case study with a parallel evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):205–225, 2010.

[139] G.B. Huang, D. Wang, and Y. Lan. Extreme learning machines: a survey. *International Journal of Machine Learning & Cybernetics*, 2(2):107–122, 2011.

[140] G.B. Huang, Q.Y. Zhu, and C.K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[141] V. L. Huang, A.K. Qin, P.N. Suganthan, and M.F. Tasgetiren. Multi-objective optimization based on self-adaptive differential evolution algorithm. In CEC-2007 [48], pages 3601–3608.

[142] V. L. Huang, S-Z Zhao, R. Mallipeddi, and P.N. Suganthan. Multi-objective optimization using self-adaptive differential evolution algorithm. In CEC-2009 [49], pages 190–194.

[143] V.L. Huang, A.K. Qin, and P.N. Suganthan. Self-adaptive differential evolution algorithm for constrained real-parameter optimization. In CEC-2006 [47], pages 17–24.

[144] *Proceedings of the First IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1994.

[145] *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1996.

[146] C. Igel, N. Hansen, and S. Roth. Covariance matrix adaptation for multi-objective optimization. *Evol. Comput.*, 15(1):1–28, 2007.

[147] A. Iorio and X. Li. Parameter control within a co-operative co-evolutionary genetic algorithm. In J.J Merelo Guervos *et al*, editor, *Proceedings of the 7th Conference on Parallel Problem Solving from Nature*, number 2439 in Lecture Notes in Computer Science, pages 247–256. Springer, Berlin, Heidelberg, New York, 2002.

[148] T. Jansen and K.A. De Jong. An analysis of the role of offspring population size in EAs. In Langdon *et al* [173], pages 238–246.

[149] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005.

[150] J.A. Joines and C.R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. In ICEC-94 [144], pages 579–584.

[151] W. Kantschik, P. Dittrich, M. Brameier, and W. Banzhaf. Meta-evolution in graph GP. In *Second European Workshop on Genetic Programming, Goteborg*, pages 15–28. Springer-Verlag, 1999.

[152] G. Karafotias, A.E. Eiben, and M. Hoogendoorn. Generic parameter control with reinforcement learning. In Dirk V. Arnold, editor, *GECCO '14: Proceedings of the 16th annual conference on Genetic and evolutionary computation*, pages 1319–1326, New York, NY, USA, 2014. ACM.

[153] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. Parameter control: strategy or luck? In Christian Blum, editor, *GECCO '13 Companion: Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 215–216, New York, NY, USA, 2013. ACM.

[154] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. Why parameter control mechanisms should be benchmarked against random variation. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, pages 349–355, Cancun, Mexico, 2013. IEEE Press.

[155] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, to appear, DOI: 10.1109/TEVC.2014.2308294, 2014.

[156] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. Evaluating reward definitions for parameter control. In A. M. Mora and G. Squillero, editors, *Applications of Evolutionary Computation*, volume 9028 of *Lecture Notes in Computer Science*, pages 667–680. Springer International Publishing, 2015.

[157] G. Karafotias, M. Hoogendoorn, and B. Weel. Comparing generic parameter controllers for EAs. In *Foundations of Computational Intelligence (FOCI), 2014 IEEE Symposium on*, pages 46–53, Dec 2014.

[158] G. Karafotias, S.K. Smit, and A.E. Eiben. A generic approach to parameter control. In C. Di Chio *et al*, editor, *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, number 7248 in Lecture Notes in Computer Science, pages 366–375. Springer, Berlin, Heidelberg, New York, 2012.

[159] G. Karafotias, S.K. Smit, and A.E. Eiben. Tuning evolutionary algorithms and tuning evolutionary algorithm controllers. In B. Filipič and J. Šilc, editors, *Proceedings of the Fifth International Conference on Bioinspired Optimization Methods, BIOMA 2012*, pages 3–21. Jožef Stefan Institute, 2012.

[160] A. Kaveh and M. Shahrouzi. Dynamic selective pressure using hybrid evolutionary and ant system strategies for structural optimization. *Int J Numer Meth Eng*, 73:544–563, 2008.

[161] S. Kazarlis and V. Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In A.E Eiben, T. Bäck, M. Schoenauer, and H.P. Schwefel, editors, *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, number 1498 in Lecture Notes in Computer Science, pages 211–220. Springer, Berlin, Heidelberg, New York, 1998.

[162] E. Kee, S. Airey, and W. Cyre. An adaptive genetic algorithm. In Spector *et al* [259], pages 391–397.

[163] M. Keijzer, editor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006)*. Morgan Kaufmann, San Francisco, 2006.

[164] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[165] V.K. Koumousis and C.P. Katsaras. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, 10(1):19–28, 2006.

[166] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[167] O. Kramer. *Self-adaptive heuristics for evolutionary computation*. PhD thesis, University of Paderborn, http://link.springer.com/book/10.1007/978-3-540-69281-2/page/1, 2008.

[168] O. Kramer. *Self-adaptive heuristics for evolutionary computation*, volume 147 of *Studies in Computational Intelligence*. Sptinger, 2008.

[169] O. Kramer. Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 3(2):51–65, 2010.

[170] N. Krasnogor et al., editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*, Dublin, Ireland, 12-16 July 2011. ACM.

[171] T. Krink, P. Rickers, and R. Thomsen. Applying self-organised criticality to evolutionary algorithms. In Schoenauer *et al* [243], pages 375–384.

[172] T. Krink and R. Thomsen. Self-organized criticality and mass extinction in evolutionary algorithms. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 2, pages 1155–1161. IEEE, 2001.

[173] W.B. Langdon *et al*, editor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*. Morgan Kaufmann, San Francisco, 9-13 July 2002.

[174] F. Lardeux and A. Goëffon. A dynamic island-based genetic algorithms framework. In *Proceedings of the 8th international conference on Simulated evolution and learning*, SEAL'10, pages 156–165. Springer-Verlag, 2010.

[175] J.L.J. Laredo, C. Fernandes, J.J. Merelo, and C. Gagné. Improving genetic algorithms performance via deterministic population shrinkage. In Franz Rothlauf, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)*, pages 819–826. ACM, 2009.

[176] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential Monte Carlo methods. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *NIPS*. Curran Associates, Inc., 2007.

[177] M.A. Lee and H. Takagi. Dynamic control of genetic algorithms using fuzzy logic techniques. In Forrest [100], pages 76–83.

[178] A. Lemonge, C.C. Afonso, and H.J.C. Barbosa. An adaptive penalty scheme for genetic algorithms in structural optimization. *International Journal for Numerical Methods in Engineering*, 59(5):703–736, 2003.

[179] K. Li, Á. Fialho, S. Kwong, and Q. Zhang. Adaptive operator selection with bandits for multiobjective evolutionary algorithm based decomposition. *Evolutionary Computation, IEEE Transactions on*, 2013 to appear.

[180] K. Li, Á. Fialho, and S. Kwong. Multi-objective differential evolution with adaptive control of parameters and operators. In C.A.Coello Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 473–487. Springer Berlin Heidelberg, 2011.

[181] N.Y. Liang, G.B. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *Neural Networks, IEEE Transactions on*, 17(6):1411–1423, 2006.

[182] T. Liao and T. Stützle. Bounding the population size of IPOP-CMA-ES on the noiseless BBOB testbed. In *Proceeding of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion*, pages 1161–1168. ACM, 2013.

[183] J. Lis and M. Lis. Self-adapting parallel genetic algorithm with the dynamic probability, crossover rate and population size. In J. Arabas, editor, *Proceedings of the First Polish Evolutionary Algorithms Conference*, pages 79–86, 1996.

[184] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005.

[185] S.-H. Liu, M. Crepinsek, M. Mernik, and A. Cardenas. Parameter control of EAs using exploration and exploitation measures: Preliminary results. In B. Filipic and J. Silc, editors, *Bioinspired Optimization Methods and their Applications, International Conference on*, pages 141–150. Jozef Stefan Institute, 2012.

[186] F.G. Lobo. Idealized dynamic population sizing for uniformly scaled problems. In Krasnogor et al. [170], pages 917–924.

[187] F.G. Lobo and C.F. Lima. Revisiting evolutionary algorithms with on-the-fly population size adjustment. In Keijzer [163], pages 1241–1248.

[188] F.G. Lobo and C.F. Lima. Adaptive population sizing schemes in genetic algorithms. In Lobo et al. [189], pages 185–204.

[189] F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, 2007.

[190] E. Chicken M. Hollander, D.A. Wolfe. *Nonparametric Statistical Methods*. Wiley, 3 edition, 2014.

[191] Y. Maeda, M. Ishita, and Q. Li. Fuzzy adaptive search method for parallel genetic algorithm with island combination process. *Int. J. Approx. Reasoning*, 41(1):59–73, 2006.

[192] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Appl. Soft Comput.*, 11(2):1679–1696, 2011.

[193] R. Männer and B. Manderick, editors. *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature*. Elsevier, 1992.

[194] J. Marin and R.V. Sole. Evolutionary optimization through extinction dynamics. In Banzhaf *et al* [27], pages 1344–1349.

[195] M.H. Maruo, H.S. Lopes, and M.R. Delgado. Self-adapting evolutionary parameters: Encoding aspects for combinatorial optimization problems. In G.. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *Lecture Notes in Computer Science*, pages 154–165. Springer Berlin Heidelberg, 2005.

[196] J. Maturana. *Controle Generique de Parametres pour les Algorithmes Evolutionnaires (General Control of Parameters for Evolutionary Algorithms)*. PhD thesis, University of Angers, http://tel.archives-ouvertes.fr/docs/00/45/91/85/PDF/These_JorgeMATURANA.pdf, 2009.

[197] J. Maturana, Á. Fialho, F. Saubion, M. Schoenauer, and M. Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In CEC-2009 [49], pages 365 –372.

[198] J. Maturana, F. Lardeux, and F. Saubion. Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, 16(6):881–909, 2010.

[199] J. Maturana and F. Saubion. Towards a generic control strategy for evolutionary algorithms: an adaptive fuzzy-learning approach. In CEC-2007 [48], pages 4546–4553.

[200] J. Maturana and F. Saubion. A compass to guide genetic algorithms. In Rudolph *et al* [233], pages 256–265.

[201] J. Maturana and F. Saubion. On the design of adaptive control strategies for evolutionary algorithms. In *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 303–315. Springer, 2008.

[202] B. McGinley, J. Maher, C. O'Riordan, and F. Morgan. Maintaining healthy population diversity using adaptive crossover, mutation, and selection. *Evolutionary Computation, IEEE Transactions on*, 15(5):692 –714, 2011.

[203] B. McGinley, F. Morgan, and C. O'Riordan. Genetic algorithms with dynamic niche sharing for multimodal function optimization. In ICEC-96 [145], pages 1127–1128.

[204] F.S. Melo and M. Lopes. Fitted natural actor-critic: A new algorithm for continuous state-action MDPs. In W. Daelemans, B. Goethals, and K. Morik, editors, *Machine*

Learning and Knowledge Discovery in Databases, volume 5212 of *Lecture Notes in Computer Science*, pages 66–81. Springer Berlin Heidelberg, 2008.

[205] S. Meyer-Nieberg and H.-G. Beyer. Self-adaptation in evolutionary algorithms. In Lobo et al. [189], pages 47–76.

[206] Z. Michalewicz and N. Attia. Evolutionary optimization of constrained problems. In A.V Sebald and L.J Fogel, editors, *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, 1994.

[207] J. del R. Millán, D. Posenato, and E. Dedieu. Continuous-action Q-learning. *Machine Learning*, 49(2-3):247–265, 2002.

[208] M. Montemurro, A. Vincenti, and P. Vannucci. The automatic dynamic penalisation method (ADP) for handling constraints with genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 256:70–87, 2012.

[209] E. Montero and M.C. Riff. Self-calibrating strategies for evolutionary approaches that solve constrained combinatorial problems. In *Proceedings of the 17th international conference on Foundations of intelligent systems*, ISMIS'08, pages 262–267. Springer-Verlag, 2008.

[210] E. Montero and M.C. Riff. On-the-fly calibrating strategies for evolutionary algorithms. *Inf. Sci.*, 181(3):552–566, February 2011.

[211] H. Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In Männer and Manderick [193], pages 15–26.

[212] S.D. Muller, N.N. Schraudolph, and P.D. Koumoutsakos. Step size adaptation in evolution strategies using reinforcement learning. In CEC-2002 [45], pages 151–156.

[213] F. Nadi and A.T. Khader. A parameter-less genetic algorithm with customized crossover and mutation operators. In Krasnogor et al. [170], pages 901–908.

[214] A. Nellis. Meta evolution. Qualifying Dissertation, 2009.

[215] T.T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24, 2012.

[216] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.

[217] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self-adaptation of evolution strategies. *Evol. Comput.*, 2(4):369–380, 1994.

[218] A. Ostermeier, A. Gawelczyk, and N. Hansen. Step-size adaption based on non-local use of selection information. In Davidor et al. [61], pages 189–198.

[219] J. Paredis. Co-evolutionary constraint satisfaction. In Davidor et al. [61], pages 46–55.

[220] I.C. Parmee, editor. *Evolutionary Design and Manufacture*. Springer, 2000.

[221] J. Pazis and M.G. Lagoudakis. Reinforcement learning in multidimensional continuous action spaces. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 97–104, 2011.

[222] M. Pelikan and J. Branke, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*. ACM, 2010.

[223] J.E. Pettinger and R.M. Everson. Controlling genetic algorithms with reinforcement learning. In Langdon *et al* [173], page 692.

[224] W. Qian and A. li. Adaptive differential evolution algorithm for multiobjective optimization problems. *Applied Mathematics and Computation*, 201(12):431 – 440, 2008.

[225] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In CEC-2005 [46], pages 1785–1791 Vol. 2.

[226] I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1973.

[227] C.R. Reeves. Using genetic algorithms with small populations. In Forrest [100], pages 92–99.

[228] G. Reynoso-Meza, J. Sanchis, X. Blasco, and M. Martinez. An empirical study on parameter selection for multiobjective optimization algorithms using differential evolution. In *Differential Evolution (SDE), 2011 IEEE Symposium on*, pages 1–7, 2011.

[229] M.C. Riff and X. Bonnaire. Inheriting parents operators: A new dynamic strategy for improving evolutionary algorithms. In M.-S. Hacid, Z. Ras, D. Zighed, and Y. Kodratoff, editors, *Foundations of Intelligent Systems*, volume 2366 of *Lecture Notes in Computer Science*, pages 333–341. Springer Berlin / Heidelberg, 2002.

[230] G. Rudolph. Self-adaptive mutations may lead to premature convergence. *Evolutionary Computation, IEEE Transactions on*, 5(4):410 –414, 2001.

[231] G. Rudolph. Evolutionary strategies. In G. Rozenberg, T. Bäck, and J.N. Kok, editors, *Handbook of Natural Computing*, pages 673–698. Springer Berlin Heidelberg, 2012.

[232] G. Rudolph and J. Sprave. A cellular genetic algorithm with selfadjusting acceptance threshold. In *Proc. 1st IEE/IEEE Int. Conf. Genetic Algorithms in Eng. Sys.: Innovations and Appl*, pages 365–372, 1995.

[233] G. Rudolph *et al*, editor. *Proceedings of the 10th Conference on Parallel Problem Solving from Nature*, volume 5199 of *Lecture Notes in Computer Science*. Springer, 2008.

[234] T.P. Runarsson *et al*, editor. *Proceedings of the 9th Conference on Parallel Problem Solving from Nature*, volume 4193 of *Lecture Notes in Computer Science*. Springer, 2006.

[235] C. Ryan and M. Keijzer, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2008)*. ACM, 2008.

[236] H. Sakanashi, K. Suzuki, and Y. Kakazui. Controlling dynamics of GA through filtered evaluation function. In Davidor et al. [61], pages 239–248.

[237] Y. Sakurai, K. Takada, T. Kawabe, and S. Tsuruta. A method to control parameters of evolutionary algorithms by using reinforcement learning. In *Signal-Image Technology and Internet-Based Systems (SITIS), 2010 Sixth International Conference on*, pages 74–79, 2010.

[238] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis. The Primer*. John Wiley & Sons, 2008.

[239] A.V. Samsonovich and K.A. De Jong. Pricing the 'free lunch' of meta-evolution. In Beyer and O'Reilly [30], pages 1355–1362.

[240] J.C. Santamaría, R.S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adapt. Behav.*, 6(2):163–217, 1998.

[241] J.D. Schaffer, R.A. Caruana, L.J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J.D Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann, San Francisco, 1989.

[242] J.D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 36–40. L. Erlbaum Associates Inc., 1987.

[243] M Schoenauer *et al*, editor. *Proceedings of the 6th Conference on Parallel Problem Solving from Nature*, number 1917 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, New York, 2000.

[244] H.-P Schwefel. *Numerical Optimisation of Computer Models*. Wiley, New York, 1981.

[245] H.-P Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[246] H.-P Schwefel and R Männer, editors. *Proceedings of the 1st Conference on Parallel Problem Solving from Nature*, number 496 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, New York, 1991.

[247] S.K. Smit. *Parameter Tuning and Scientific Testing in Evolutionary Algorithms*. PhD thesis, VU University Amsterdam, Department of Computer Sciences, http://www.cs.vu.nl/en/Images/Selmar%20Smit17okt12_tcm75-310264.pdf, October 2012.

[248] S.K. Smit and A.E. Eiben. Multi-problem parameter tuning using BONESA. In JK Hao, P Legrand, P Collet, N Monmarché, E Lutton, and M Schoenauer, editors, *Artificial Evolution, 10th International Conference Evolution Artificielle*, pages 222–233, 2011.

[249] S.K. Smit and A.E. Eiben. Population diversity index: A new measure for population diversity. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2011.

[250] A.E. Smith and D.W. Coit. Penalty functions. In *Handbook on Evolutionary Computation*, page C5.2. Oxford University Press, 1997.

[251] J.E. Smith. *Self Adaptation in Evolutionary Algorithms*. PhD thesis, University of West of England, Bristol, http://www.bit.uwe.ac.uk/j̃smith/pdfs/thesis.pdf, 1998.

[252] J.E. Smith and T. C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 1(2):81–87, 1997.

[253] J.E. Smith and T.C. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In ICEC-96 [145], pages 318–323.

[254] R.E. Smith and E. Smuda. Adaptively resizing populations: Algorithm, analysis, and first results. *Complex Systems*, 9(1):47–72, 1995.

[255] E. Smorodkina and D. Tauritz. Toward automating EA configuration: The parent selection stage. In CEC-2007 [48], pages 63–70.

[256] E. Smorodkina and D.R. Tauritz. Greedy population sizing for evolutionary algorithms. In CEC-2007 [48], pages 2181–2187.

[257] M.T.J. Spaan. Partially observable markov decision processes. In Wiering and van Otterlo [287], pages 387–414.

[258] W.M. Spears. Adapting crossover in evolutionary algorithms. In *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384. MIT Press, 1995.

[259] L. Spector *et al*, editor. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. Morgan Kaufmann, San Francisco, 2001.

[260] M. Srinivas and L.M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, 1994.

[261] S.A. Stanhope and J.M. Daida. Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In V.W Porto, N Saravanan, D Waagen, and A.E Eiben, editors, *Proceedings of the 7th Annual Conference on Evolutionary*

Programming, number 1477 in LNCS, pages 693–702. Springer, Berlin, Heidelberg, New York, 1998.

[262] R. Storn and K. Price. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[263] R.S. Sutton and A.G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.

[264] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing*, 10(8):673–686, 2006.

[265] B. Tessema and G.G. Yen. A self adaptive penalty function based algorithm for constrained optimization. In CEC-2006 [47], pages 246–253.

[266] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In Beyer and O'Reilly [30], pages 1539–1546.

[267] D. Thierens. Adaptive strategies for operator allocation. In Lobo et al. [189], pages 77–90.

[268] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[269] A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evol. Comput.*, 6(2):161–184, 1998.

[270] R.K. Ursem. *Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization*. PhD thesis, University of Aarhus, http://www.brics.dk/DS/03/6/BRICS-DS-03-6.pdf, 2003.

[271] W.T. B. Uther and M.M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 769–774. American Association for Artificial Intelligence, 1998.

[272] F. Vafaee and P.C. Nelson. An explorative and exploitative mutation scheme. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 1 –8, Barcelona, Spain, 2010. IEEE Computational Intelligence Society, IEEE Press.

[273] H. van Hasselt. Reinforcement learning in continuous state and action spaces. In Wiering and van Otterlo [287], pages 207–251.

[274] H. van Hasselt and M.A. Wiering. Reinforcement learning in continuous action spaces. In *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*, pages 272–279, 2007.

[275] H. van Hasselt and M.A. Wiering. Using continuous action spaces to solve discrete problems. In *International Joint Conference on Neural Networks, IJCNN 2009, Atlanta, Georgia, USA, 14-19 June 2009*, pages 1149–1156. IEEE Computer Society, 2009.

[276] M. van Otterlo and M. Wiering. Reinforcement learning and Markov Decision Processes. In Wiering and van Otterlo [287], pages 3–42.

[277] M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3):35:1–35:33, 2013.

[278] N. Veerapen, J. Maturana, and F. Saubion. A comparison of operator utility measures for on-line operator selection in local search. In *Proceedings of the 6th international conference on Learning and Intelligent Optimization*, LION'12, pages 497–502. Springer-Verlag, 2012.

[279] N. Wagner and Z. Michalewicz. Genetic programming with efficient population control for financial time series prediction. In E.D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 458–462, 2001.

[280] N. Wagner and Z. Michalewicz. Parameter adaptation for GP forecasting applications. In Lobo et al. [189], pages 295–309.

[281] N. Wagner, Z. Michalewicz, M. Khouja, and R. McGregor. Time series forecasting for dynamic environments: The dyfor genetic program model. *IEEE Transactions on Evolutionary Computing*, 11(4):433–452, 2007.

[282] R.J. Wang, Y. Ru, and Q. Long. Improved adaptive and multi-group parallel genetic algorithm based on good-point set. *Journal of Software*, 4(4):348–356, 2009.

[283] Y. Wang, Z. Cai, Y. Zhou, and Z. Fan. Constrained optimization based on hybrid evolutionary algorithm and adaptive constraint-handling technique. *Structural and Multidisciplinary Optimization*, 37(4):395–413, 2009.

[284] R.A. Watson, S.G. Ficici, and J.B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, April 2002.

[285] J.M. Whitacre, T.Q. Pham, and R.A. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. In Keijzer [163], pages 1345–1352.

[286] W. Wickramasinghe, M. van Steen, and A.E. Eiben. Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In D. Thierens *et al.*, editor, *GECCO '07: Proc of the 9th conference on Genetic and Evolutionary Computation*, pages 1460–1467. ACM Press, 2007.

[287] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*. Springer Berlin Heidelberg, 2012.

[288] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, Apr 1997.

[289] Y.-Y. Wong, K.-H. Lee, K.-S. Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 7(8):506–515, 2003.

[290] D. Xie, L. Ding, S. Wang, Z. Guo, Y. Hu, and C. Xie. Self-adaptive differential evolution based multi-objective optimization incorporating local search and indicator-based selection. In D-S. Huang, C. Jiang, V. Bevilacqua, and J.C. Figueroa, editors, *Intelligent Computing Technology*, volume 7389 of *Lecture Notes in Computer Science*, pages 25–33. Springer Berlin Heidelberg, 2012.

[291] S. Yang. On the design of diploid genetic algorithms for problem optimization in dynamic environments. In CEC-2006 [47], pages 1362–1369.

[292] S. Yang, T.T. Nguyen, and C. Li. Evolutionary dynamic optimization: Test and evaluation environments. In Yang and Yao [294], pages 3–37.

[293] S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Comput.*, 9(11):815–834, 2005.

[294] S. Yang and X. Yao, editors. *Evolutionary Computation for Dynamic Optimization Problems*. Springer-Verlag Berlin Heidelberg, 2013.

[295] X. Yao *et al*, editor. *Proceedings of the 8th Conference on Parallel Problem Solving from Nature*, volume 3242 of *Lecture Notes in Computer Science*. Springer, 2004.

[296] T.-L. Yu, D.E. Goldberg, A. Yassine, and Y.-P. Chen. Genetic algorithm design inspired by organizational theory: pilot study of a dependency structure matrix driven genetic algorithm. In E. Cantú-Paz *et al*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, volume 2723 of *Lecture Notes in Computer Science*, pages 1620–1621. Springer, 2003.

[297] T.-L. Yu, K. Sastry, and D.E. Goldberg. Online population size adjusting using noise and substructural measurements. In CEC-2005 [46], pages 2491–2498.

[298] T.-L. Yu, K. Sastry, and D.E. Goldberg. Population sizing to go: Online adaptation using noise and substructural measurements. In Lobo et al. [189], pages 205–223.

[299] Z.H. Zhan and J. Zhang. Co-evolutionary differential evolution with dynamic population size and adaptive migration strategy. In Krasnogor et al. [170], pages 211–212.

[300] J. Zhang and A.C. Sanderson. Jade: Self-adaptive differential evolution with fast and reliable convergence performance. In CEC-2007 [48], pages 2251–2258.

[301] J. Zhang and A.C. Sanderson. JADE: Adaptive differential evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on*, 13(5):945–958, 2009.

[302] K. Zielinski and R. Laur. Parameter adaptation for differential evolution with design of experiments. In B. Kovalerchuk, editor, *Proceedings of the Second IASTED International Conference on Computational Intelligence, San Francisco, California, USA, November 20-22, 2006*, pages 216–221. IASTED/ACTA Press, 2006.

[303] K. Zielinski and R. Laur. Differential evolution with adaptive parameter setting for multi-objective optimization. In CEC-2007 [48], pages 3585–3592.

[304] K. Zielinski, X. Wang, and R. Laur. Comparison of adaptive approaches for differential evolution. In Rudolph *et al* [233], pages 641–650.

[305] K. Zielinski, P. Weitkemper, R. Laur, and K.-D. Kammeyer. Parameter study for differential evolution using a power allocation problem including interference cancellation. In CEC-2006 [47], pages 1857–1864.

Summary

The research described in this thesis investigates automatic on-line control of the parameters of evolutionary algorithms (EAs).

Evolutionary algorithms are a class of search and optimisation algorithms based on the paradigm of biological evolution in nature. EAs are used to find solutions for difficult problems, i.e. problems of high complexity, high dimensionality and very large search spaces. EAs are population-based algorithms: they maintain a population of possible solutions at all times. Possible (candidate) solutions to a problem are encoded in an appropriate representation that contains all the necessary information. Candidate solutions are combined with each other and are randomly perturbed, a process that corresponds to reproduction in biological organisms. Each candidate solution is assigned a fitness value that expresses how well it solves the problem at hand. Based on these fitness values, candidate solutions compete for a place in the population and a chance for mating; this resembles the selection that organisms go through in nature and the concept of the survival of the fittest. Though EAs are very far from accurately simulating biological evolution, they borrow the basic notions of it and use them to successfully find good solutions to problems.

The core process of an EA consists of a loop with a number of steps involving the creation and selection of candidate solutions. Each such step in the loop may include several parameters, either numeric or nominal. The effectiveness of an EA is very often influenced to a great extent by the specific values of these parameters. Therefore, successfully applying an EA often becomes a matter of properly setting its parameter values. To further complicate things, it has also been shown that changing the values of some parameters during different stages of a single run can also have an impact on the performance of an EA.

This poses a challenge: how to set the value of a parameter at any given time during a run so as to maximize the effectiveness of the EA? Parameter control attempts to meet that challenge with the use of mechanisms that vary the parameters of an EA on-the-fly

according to an underlying strategy. A parameter controller adds an extra step to the core loop of an EA: at every iteration the controller uses information extracted from the current population of the EA to make a decision about how to subsequently set the values of the parameters of the EA.

However, though the principle is there, parameter control is, practically, still an open problem as there are no standard and widely used solutions. The purpose of this thesis is to contribute to making parameter control more applicable and relevant in practice by (i) providing a theoretical basis that identifies the key elements of parameter control and helps with designing new controllers, and (ii) working towards a one-fits-all solution, i.e. a controller in the form of a readily available plugin that can be applied to any EA.

The text is divided into two parts. The first part examines the basic notion of parameter control and its relation to tuning and the overall parameter setting problem, suggests a framework for parameter control from a component viewpoint as well as in the context of Reinforcement Learning, and presents an extensive literature review of the field of parameter control identifying key trends and challenges. Based on that, the second part presents a number of parameter controllers designed as independent components that can be used as generic plugins for any conventional EA, and experimentally evaluates these control designs with several EA and problem combinations, including real world and dynamic problems. Experimental results show that a generally applicable parameter controller is a viable idea that could be mostly beneficial when controlling evolutionary algorithms that are not extensively refined. This can be especially useful when applying newly developed and ad hoc evolutionary algorithms as is often the case in industrial and commercial environments.

Samenvatting

Het onderzoek beschreven in dit proefschrift onderzoekt automatische online controle van de parameters van evolutionaire algoritmes (EAs)

Evolutionaire algoritmes zijn een klasse van zoek- en optimalisatie-algoritmes op basis van het paradigma van biologische evolutie in de natuur. EAs worden gebruikt om oplossingen voor moeilijke problemen te vinden, namelijk de problemen van hoge complexiteit, hoge dimensionaliteit en met een zeer grote zoekruimte. EAs zijn algoritmes op basis van populaties, dat wil zeggen dat ze te allen tijde een populatie van mogelijke oplossingen bijhouden. Mogelijk (kandidaat-) oplossingen voor een probleem zijn gecodeerd in een passende representatie die alle benodige informatie bevat. Kandidaat- oplossingen worden met elkaar gecombineerd en willekeurig veranderd, een proces dat overeenkomt met voortplanting in biologische organismen. Elke kandidaat-oplossing krijgt een fitness-waarde die uitdrukt hoe goed deze het probleem oplost. Op basis van deze fitness waarden, moeten kandidaat oplossingen strijden voor een plaats in de populatie en een kans om te paren; dit lijkt op de selectie die organismen ondergaan in de natuur en het concept van de “survival of the fittest”. Hoewel EAs niet de biologische evolutie exact simuleren, lenen ze de basisbegrippen ervan en worden ze gebruikt om succesvol goede oplossingen voor problemen te vinden.

Het kernproces een EAs bestaat uit het herhaaldelijk uitvoeren van een aantal stappen, inclusief het initialiseren en selecteren van kandidaat-oplossingen. Tijdens het uitvoeren van deze stappen dienen een aantal parameters van een waarde te worden voorzien. Hierbij kunnen de toegestane waardes per parameter heel erg verschillend van aard zijn, ze zijn echter altijd numeriek of nominaal. De effectiviteit van een EA wordt vaak sterk benvloed door de specifieke waarden van deze parameters. Daarom wordt het met succes toepassen van een EA vaak een kwestie van de juiste instelling van deze parameters vinden. Om de zaken verder te compliceren is ook aangetoond dat het veranderen van de waarden van

bepaalde parameters in verschillende stadia van de executie ook invloed kan hebben op de prestaties van een EA.

Dit vormt een uitdaging: hoe moeten we de waarde van een parameter instellen zodat op ieder moment tijdens een executie van het algoritme de effectiviteit van het EA gemaximaliseerd wordt? “Parameter controle” doet een poging om deze uitdaging aan te gaan door het gebruik van mechanismen die de parameters van een EA on-the-fly variren volgens een bepaalde strategie. Een parameter controle mechanisme voegt een extra stap aan toe aan de stappen van een EA: bij elke iteratie gebruikt het controle mechanisme gegevens uit de huidige populatie van het EA om een beslissing te nemen over de parameters van het EA.

Echter, hoewel het principe van parameter controle vaker voorgesteld en gemplementeerd is, is er nog steeds een open probleem want er zijn geen standaard oplossingen die overal gebruikt kunnen worden. Het doel van dit proefschrift is om bij te dragen tot het maken van parameter controle mechanismen die meer bruikbaar en relevant in de praktijk zijn door (i) het verschaffen van een theoretische basis die de belangrijkste elementen van een parameter controle mechanisme identificeert en helpt bij het ontwerpen van nieuwe controllers, en (ii) te werken aan een uniforme oplossing, te weten een controller in de vorm van een gemakkelijk toegankelijke plug-in die kan worden toegepast op elk EA.

De tekst is van dit proefschrift opgedeeld in twee delen. Het eerste deel gaat in op de fundamentele notie van parameter controle, de relatie met het optimaliseren van parameters en het probleem van parameters instellen in het algemeen. Het stelt een kader op voor de parameter controle gezien als extra component van EAs en kijkt daar met name naar in de context van Reinforcement Learning. Een uitgebreid literatuuronderzoek wordt gepresenteerd op het gebied van parameter controle, daarbij kijkend naar de belangrijkste trends en uitdagingen. Op basis hiervan bevat het tweede deel een aantal parameter controle mechanismen ontworpen als onafhankelijke componenten. Deze kunnen worden gebruikt als generieke plug-ins bij conventionele EAs. Ze worden experimenteel gevalueerd door ze te combineren met verschillende EAs en problemen, waaronder realistische dynamische problemen. De experimentele resultaten van dit onderzoek demonstreren dat een generiek ontwerp van een parameter controle mechanisme van toegevoegde waarde is, vooral voor niet volledig geoptimaliseerde evolutionaire algoritmen. Dit kan vooral nuttig zijn bij nieuwe en ad hoc gecreerde evolutionaire algoritmen zoals vaak het geval is in industrile en commercile omgevingen.

SIKS Dissertatiereeks

2009

2009-01 Rasa Jurgelenaite (RUN) Symmetric Causal Independence Models
2009-02 Willem Robert van Hage (VU) Evaluating Ontology-Alignment Techniques
2009-03 Hans Stol (UvT) A Framework for Evidence-based Policy Making Using IT
2009-04 Josephine Nabukenya (RUN) Improving the Quality of Organisational Policy Making using Collaboration Engineering
2009-05 Sietse Overbeek (RUN) Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
2009-06 Muhammad Subianto (UU) Understanding Classification
2009-07 Ronald Poppe (UT) Discriminative Vision-Based Recovery and Recognition of Human Motion
2009-08 Volker Nannen (VU) Evolutionary Agent-Based Policy Analysis in Dynamic Environments
2009-09 Benjamin Kanagwa (RUN) Design, Discovery and Construction of Service-oriented Systems
2009-10 Jan Wielemaker (UVA) Logic programming for knowledge-intensive interactive applications
2009-11 Alexander Boer (UVA) Legal Theory, Sources of Law & the Semantic Web
2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin) Operating Guidelines for Services
2009-13 Steven de Jong (UM) Fairness in Multi-Agent Systems
2009-14 Maksym Korotkiy (VU) From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
2009-15 Rinke Hoekstra (UVA) Ontology Representation - Design Patterns and Ontologies that Make Sense
2009-16 Fritz Reul (UvT) New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT) Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI) Armada, An Evolving Database System
2009-19 Valentin Robu (CWI) Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
2009-20 Bob van der Vecht (UU) Adjustable Autonomy: Controling Influences on Decision Making
2009-21 Stijn Vanderlooy (UM) Ranking and Reliable Classification
2009-22 Pavel Serdyukov (UT) Search For Expertise: Going beyond direct evidence
2009-23 Peter Hofgesang (VU) Modelling Web Usage in a Changing Environment
2009-24 Annerieke Heuvelink (VUA) Cognitive Models for Training Simulations
2009-25 Alex van Ballegooij (CWI) ”RAM: Array Database Management through Relational Mapping”
2009-26 Fernando Koch (UU) An Agent-Based Model for the Development of Intelligent Mobile Services
2009-27 Christian Glahn (OU) Contextual Support of social Engagement and Reflection on the Web
2009-28 Sander Evers (UT) Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT) Model-Driven Semantic Integration of Service-Oriented Applications
2009-30 Marcin Zukowski (CWI) Balancing vectorized query execution with bandwidth-optimized storage
2009-31 Sofiya Katrenko (UVA) A Closer Look at Learning Relations from Text
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU) Architectural Knowledge Management: Supporting Architects and Auditors
2009-33 Khiet Truong (UT) How Does Real Affect Affect Affect Recognition In Speech?
2009-34 Inge van de Weerd (UU) Advancing in Software Product Management: An Incremental Method Engineering Approach
2009-35 Wouter Koelewijn (UL) Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
2009-36 Marco Kalz (OUN) Placement Support for Learners in Learning Networks
2009-37 Hendrik Drachsler (OUN) Navigation Support for Learners in Informal Learning Networks
2009-38 Riina Vuorikari (OU) Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) Service Substitution – A Behavioral Approach Based on Petri Nets
2009-40 Stephan Raaijmakers (UvT) Multinomial Language Learning: Investigations into the Geometry of Language
2009-41 Igor Berezhnyy (UvT) Digital Analysis of Paintings
2009-42 Toine Bogers (UvT) Recommender Systems for Social Bookmarking
2009-43 Virginia Nunes Leal Franqueira (UT) Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
2009-44 Roberto Santana Tapia (UT) Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU) Making Pattern Mining Useful
2009-46 Loredana Afanasiev (UvA) Querying XML: Benchmarks and Recursion

2010

2010-01 Matthijs van Leeuwen (UU) Patterns that Matter
2010-02 Ingo Wassink (UT) Work flows in Life Science
2010-03 Joost Geurts (CWI) A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT) Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments
2010-05 Claudia Hauff (UT) Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT) Rapid Adaptation of Video Game AI
2010-07 Wim Fikkert (UT) Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL) Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments
2010-09 Hugo Kielman (UL) A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging
2010-10 Rebecca Ong (UL) Mobile Communication and Protection of Children
2010-11 Adriaan Ter Mors (TUD) The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UU) Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN) High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU) Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT) Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD) Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotoulas (VU) Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU) Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriette Cramer (UvA) People's Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT) Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT) Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI) End-user Support for Access to Heterogeneous Linked Data
2010-23 Bas Steunebrink (UU) The Logical Structure of Emotions
2010-24 Dmytro Tykhonov Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU) Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI) XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL) Automatisch contracteren
2010-28 Arne Koopman (UU) Characteristic Relational Patterns
2010-29 Stratos Idreos(CWI) Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marieke van Erp (UvT) Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UVA) Ontology Enrichment from Heterogeneous Sources on the Web
2010-32 Marcel Hiel (UvT) An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT) Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval
2010-34 Teduh Dirgahayu (UT) Interaction Design in Service Compositions
2010-35 Dolf Trieschnigg (UT) Proof of Concept: Concept-based Biomedical Information Retrieval
2010-36 Jose Janssen (OU) Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification
2010-37 Niels Lohmann (TUE) Correctness of services and their composition
2010-38 Dirk Fahland (TUE) From Scenarios to components
2010-39 Ghazanfar Farooq Siddiqui (VU) Integrative modeling of emotions in virtual agents
2010-40 Mark van Assem (VU) Converting and Integrating Vocabularies for the Semantic Web
2010-41 Guillaume Chaslot (UM) Monte-Carlo Tree Search
2010-42 Sybren de Kinderen (VU) Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU) A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE) An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT) A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU) e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT) Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Withdrawn

2010-49 Jahn-Takeshi Saito (UM) Solving difficult game positions

2010-50 Bouke Huurnink (UVA) Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI) Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU) Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA) Combining Concepts and Language Models for Information Access

2011

2011-01 Botond Cseke (RUN) Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU) Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE) Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU) Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU) Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE) Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT) Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU) BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU) Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT) Cloud Content Contention

2011-11 Dhaval Vyas (UT) Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE) Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT) Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA) The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM) Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA) Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM) Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU) The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU) Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD) Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA) System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA) Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT) Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)) Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU) Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU) Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein(UVA) Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE) Discrimination-aware Classification

2011-30 Egon van den Broek (UT) Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR) Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
2011-32 Nees-Jan van Eck (EUR) Methodological Advances in Bibliometric Mapping of Science
2011-33 Tom van der Weide (UU) Arguing to Motivate Decisions
2011-34 Paolo Turrini (UU) Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
2011-35 Maaike Harbers (UU) Explaining Agent Behavior in Virtual Training
2011-36 Erik van der Spek (UU) Experiments in serious game design: a cognitive approach
2011-37 Adriana Burlutiu (RUN) Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
2011-38 Nyree Lemmens (UM) Bee-inspired Distributed Optimization
2011-39 Joost Westra (UU) Organizing Adaptation using Agents in Serious Games
2011-40 Viktor Clerc (VU) Architectural Knowledge Management in Global Software Development
2011-41 Luan Ibraimi (UT) Cryptographically Enforced Distributed Data Access Control
2011-42 Michal Sindlar (UU) Explaining Behavior through Mental State Attribution
2011-43 Henk van der Schuur (UU) Process Improvement through Software Operation Knowledge
2011-44 Boris Reuderink (UT) Robust Brain-Computer Interfaces
2011-45 Herman Stehouwer (UvT) Statistical Language Models for Alternative Sequence Selection
2011-46 Beibei Hu (TUD) Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
2011-47 Azizi Bin Ab Aziz(VU) Exploring Computational Models for Intelligent Support of Persons with Depression
2011-48 Mark Ter Maat (UT) Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
2011-49 Andreea Niculescu (UT) Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

2012-01 Terry Kakeeto (UvT) Relationship Marketing for SMEs in Uganda
2012-02 Muhammad Umair(VU) Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
2012-03 Adam Vanya (VU) Supporting Architecture Evolution by Mining Software Repositories
2012-04 Jurriaan Souer (UU) Development of Content Management System-based Web Applications
2012-05 Marijn Plomp (UU) Maturing Interorganisational Information Systems
2012-06 Wolfgang Reinhardt (OU) Awareness Support for Knowledge Workers in Research Networks
2012-07 Rianne van Lambalgen (VU) When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
2012-08 Gerben de Vries (UVA) Kernel Methods for Vessel Trajectories
2012-09 Ricardo Neisse (UT) Trust and Privacy Management Support for Context-Aware Service Platforms
2012-10 David Smits (TUE) Towards a Generic Distributed Adaptive Hypermedia Environment
2012-11 J.C.B. Rantham Prabhakara (TUE) Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
2012-12 Kees van der Sluijs (TUE) Model Driven Design and Data Integration in Semantic Web Information Systems
2012-13 Suleman Shahid (UvT) Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
2012-14 Evgeny Knutov(TUE) Generic Adaptation Framework for Unifying Adaptive Web-based Systems
2012-15 Natalie van der Wal (VU) Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
2012-16 Fiemke Both (VU) Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
2012-17 Amal Elgammal (UvT) Towards a Comprehensive Framework for Business Process Compliance
2012-18 Eltjo Poort (VU) Improving Solution Architecting Practices
2012-19 Helen Schonenberg (TUE) What's Next? Operational Support for Business Process Execution
2012-20 Ali Bahramisharif (RUN) Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
2012-21 Roberto Cornacchia (TUD) Querying Sparse Matrices for Information Retrieval
2012-22 Thijs Vis (UvT) Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT) Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT) Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT) Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA) Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT) Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT) Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT) Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD) Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN) A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD) Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN) Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN) Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU) Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN) Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN) A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU) Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT) Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT) Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU) Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU) Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn

2012-44 Anna Tordai (VU) On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT) A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA) Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA) Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE) Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM) Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD) Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD) Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013

2013-01 Viorel Milea (EUR) News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI) MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU) Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati(TUD) Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT) Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves(CWI) The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT) Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk(VU) Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN) Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige(UvT) A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras(TUD) Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian(VU) Knowledge-driven Migration to Services

2013-13 Mohammad Safiri(UT) Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
2013-14 Jafar Tanha (UVA) Ensemble Approaches to Semi-Supervised Learning Learning
2013-15 Daniel Hennes (UM) Multiagent Learning - Dynamic Games and Applications
2013-16 Eric Kok (UU) Exploring the practical benefits of argumentation in multi-agent deliberation
2013-17 Koen Kok (VU) The PowerMatcher: Smart Coordination for the Smart Electricity Grid
2013-18 Jeroen Janssens (UvT) Outlier Selection and One-Class Classification
2013-19 Renze Steenhuizen (TUD) Coordinated Multi-Agent Planning and Scheduling
2013-20 Katja Hofmann (UvA) Fast and Reliable Online Learning to Rank for Information Retrieval
2013-21 Sander Wubben (UvT) Text-to-text generation by monolingual machine translation
2013-22 Tom Claassen (RUN) Causal Discovery and Logic
2013-23 Patricio de Alencar Silva(UvT) Value Activity Monitoring
2013-24 Haitham Bou Ammar (UM) Automated Transfer in Reinforcement Learning
2013-25 Agnieszka Anna Latoszek-Berendsen (UM) Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
2013-26 Alireza Zarghami (UT) Architectural Support for Dynamic Homecare Service Provisioning
2013-27 Mohammad Huq (UT) Inference-based Framework Managing Data Provenance
2013-28 Frans van der Sluis (UT) When Complexity becomes Interesting: An Inquiry into the Information eXperience
2013-29 Iwan de Kok (UT) Listening Heads
2013-30 Joyce Nakatumba (TUE) Resource-Aware Business Process Management: Analysis and Support
2013-31 Dinh Khoa Nguyen (UvT) Blueprint Model and Language for Engineering Cloud Applications
2013-32 Kamakshi Rajagopal (OUN) Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
2013-33 Qi Gao (TUD) User Modeling and Personalization in the Microblogging Sphere
2013-34 Kien Tjin-Kam-Jet (UT) Distributed Deep Web Search
2013-35 Abdallah El Ali (UvA) Minimal Mobile Human Computer Interaction Promotor: Prof. dr. L. Hardman (CWI/UVA)
2013-36 Than Lam Hoang (TUe) Pattern Mining in Data Streams
2013-37 Dirk Börner (OUN) Ambient Learning Displays
2013-38 Eelco den Heijer (VU) Autonomous Evolutionary Art
2013-39 Joop de Jong (TUD) A Method for Enterprise Ontology based Design of Enterprise Information Systems
2013-40 Pim Nijssen (UM) Monte-Carlo Tree Search for Multi-Player Games
2013-41 Jochem Liem (UVA) Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
2013-42 Léon Planken (TUD) Algorithms for Simple Temporal Reasoning
2013-43 Marc Bron (UVA) Exploration and Contextualization through Interaction and Concepts

2014

2014-01 Nicola Barile (UU) Studies in Learning Monotone Models from Data
2014-02 Fiona Tuliyano (RUN) Combining System Dynamics with a Domain Modeling Method
2014-03 Sergio Raul Duarte Torres (UT) Information Retrieval for Children: Search Behavior and Solutions
2014-04 Hanna Jochmann-Mannak (UT) Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
2014-05 Jurriaan van Reijsen (UU) Knowledge Perspectives on Advancing Dynamic Capability
2014-06 Damian Tamburri (VU) Supporting Networked Software Development
2014-07 Arya Adriansyah (TUE) Aligning Observed and Modeled Behavior
2014-08 Samur Araujo (TUD) Data Integration over Distributed and Heterogeneous Data Endpoints
2014-09 Philip Jackson (UvT) Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
2014-10 Ivan Salvador Razo Zapata (VU) Service Value Networks

2014-11 Janneke van der Zwaan (TUD) An Empathic Virtual Buddy for Social Support

2014-12 Willem van Willigen (VU) Look Ma, No Hands: Aspects of Autonomous Vehicle Control

2014-13 Arlette van Wissen (VU) Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

2014-14 Yangyang Shi (TUD) Language Models With Meta-information

2014-15 Natalya Mogles (VU) Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

2014-16 Krystyna Milian (VU) Supporting trial recruitment and design by automatically interpreting eligibility criteria

2014-17 Kathrin Dentler (VU) Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

2014-18 Mattijs Ghijsen (UVA) Methods and Models for the Design and Study of Dynamic Agent Organizations

2014-19 Vinicius Ramos (TUE) Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

2014-20 Mena Habib (UT) Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

2014-21 Kassidy Clark (TUD) Negotiation and Monitoring in Open Environments

2014-22 Marieke Peeters (UU) Personalized Educational Games - Developing agent-supported scenario-based training

2014-23 Eleftherios Sidirourgos (UvA/CWI) Space Efficient Indexes for the Big Data Era

2014-24 Davide Ceolin (VU) Trusting Semi-structured Web Data

2014-25 Martijn Lappenschaar (RUN) New network models for the analysis of disease interaction

2014-26 Tim Baarslag (TUD) What to Bid and When to Stop

2014-27 Rui Jorge Almeida (EUR) Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty

2014-28 Anna Chmielowiec (VU) Decentralized k-Clique Matching

2014-29 Jaap Kabbedijk (UU) Variability in Multi-Tenant Enterprise Software

2014-30 Peter de Cock (UvT) Anticipating Criminal Behaviour

2014-31 Leo van Moergestel (UU) Agent Technology in Agile Multiparallel Manufacturing and Product Support

2014-32 Naser Ayat (UvA) On Entity Resolution in Probabilistic Data

2014-33 Tesfa Tegegne (RUN) Service Discovery in eHealth

2014-34 Christina Manteli(VU) The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.

2014-35 Joost van Ooijen (UU) Cognitive Agents in Virtual Worlds: A Middleware Design Approach

2014-36 Joos Buijs (TUE) Flexible Evolutionary Algorithms for Mining Structured Process Models

2014-37 Maral Dadvar (UT) Experts and Machines United Against Cyberbullying

2014-38 Danny Plass-Oude Bos (UT) Making brain-computer interfaces better: improving usability through post-processing.

2014-39 Jasmina Maric (UvT) Web Communities, Immigration, and Social Capital

2014-40 Walter Omona (RUN) A Framework for Knowledge Management Using ICT in Higher Education

2014-41 Frederic Hogenboom (EUR) Automated Detection of Financial Events in News Text

2014-42 Carsten Eijckhof (CWI/TUD) Contextual Multidimensional Relevance Models

2014-43 Kevin Vlaanderen (UU) Supporting Process Improvement using Method Increments

2014-44 Paulien Meesters (UvT) Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

2014-45 Birgit Schmitz (OUN) Mobile Games for Learning: A Pattern-Based Approach

2014-46 Ke Tao (TUD) Social Web Data Analytics: Relevance, Redundancy, Diversity

2014-47 Shangsong Liang (UVA) Fusion and Diversification in Information Retrieval

2015

2015-01 Niels Netten (UvA) Machine Learning for Relevance of Information in Crisis Response

2015-02 Faiza Bukhsh (UvT) Smart auditing: Innovative Compliance Checking in Customs Controls

2015-03 Twan van Laarhoven (RUN) Machine learning for network data

2015-04 Howard Spoelstra (OUN) Collaborations in Open Learning Environments

2015-05 Christoph Bösch(UT) Cryptographically Enforced Search Pattern Hiding

2015-06 Farideh Heidari (TUD) Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
2015-07 Maria-Hendrike Peetz(UvA) Time-Aware Online Reputation Analysis
2015-08 Jie Jiang (TUD) Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
2015-09 Randy Klaassen(UT) HCI Perspectives on Behavior Change Support Systems
2015-10 Henry Hermans (OUN) OpenU: design of an integrated system to support lifelong learning
2015-11 Yongming Luo(TUE) Designing algorithms for big graph datasets: A study of computing bisimulation and joins
2015-12 Julie M. Birkholz (VU) Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
2015-13 Giuseppe Procaccianti(VU) Energy-Efficient Software
2015-14 Bart van Straalen (UT) A cognitive approach to modeling bad news conversations
2015-15 Klaas Andries de Graaf (VU) Ontology-based Software Architecture Documentation
2015-16 Changyun Wei (UT) Cognitive Coordination for Cooperative Multi-Robot Teamwork
2015-17 André van Cleeff (UT) Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
2015-18 Holger Pirk (CWI) Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
2015-19 Bernardo Tabuenca (OUN) Ubiquitous Technology for Lifelong Learners
2015-20 Loïs Vanhée(UU) Using Culture and Values to Support Flexible Coordination
2015-21 Sibren Fetter (OUN) Using Peer-Support to Expand and Stabilize Online Learning
015-22 Zhemin Zhu(UT) Co-occurrence Rate Networks
2015-23 Luit Gazendam (VU) Cataloguer Support in Cultural Heritage
2015-24 Richard Berendsen (UVA) Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
2015-25 Steven Woudenberg (UU) Bayesian Tools for Early Disease Detection
2015-26 Alexander Hogenboom (EUR) Sentiment Analysis of Text Guided by Semantics and Structure
2015-27 Sndor Hman (CWI) Updating compressed column-stores
2015-28 Janet Bagorogoza (TiU) Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
2015-29 Hendrik Baier (UM) Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
2015-30 Kiavash Bahreini (OUN) Real-time Multimodal Emotion Recognition in E-Learning
2015-31 Yakup Ko (TUD) On Robustness of Power Grids
2015-32 Jerome Gard (UL) Corporate Venture Management in SMEs
2015-33 Frederik Schadd (UM) Ontology Mapping with Auxiliary Resources
2015-34 Victor de Graaff (UT) Geosocial Recommender Systems
2015-35 Junchao Xu (TUD) Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016

2016-01 Syed Saiden Abbas (RUN) Recognition of Shapes by Humans and Machines
2016-02 Michiel Christiaan Meulendijk (UU) Optimizing medication reviews through decision support: prescribing a better pill to swallow
2016-03 Maya Sappelli (RUN) Knowledge Work in Context: User Centered Knowledge Worker Support
2016-04 Laurens Rietveld (VU) Publishing and Consuming Linked Data
2016-05 Evgeny Sherkhonov (UVA) Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
2016-06 Michel Wilson (TUD) Robust scheduling in an uncertain environment
2016-07 Jeroen de Man (VU) Measuring and modeling negative emotions for virtual training
2016-08 Matje van de Camp (TiU) A Link to the Past: Constructing Historical Social Networks from Unstructured Data
2016-09 Archana Nottamkandath (VU) Trusting Crowdsourced Information on Cultural Artefacts